TECTONIC SMOOTHING AND MAPPING

A Thesis
Presented to
The Academic Faculty

by

Kai Ni

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing
College of Computing

Georgia Institute of Technology
August 2011

TECTONIC SMOOTHING AND MAPPING

Approved by:

Professor Frank Dellaert, Advisor
School of Interactive Computing
College of Computing
Georgia Institute of Technology

Professor Irfan Essa
School of Interactive Computing
College of Computing
Georgia Institute of Technology

Professor Marc Pollefeys
Department of Computer Science
ETH Zurich

Professor Henrik I. Christensen
School of Interactive Computing
College of Computing
Georgia Institute of Technology

Professor Eric N. Johnson
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 21 April 2011

to my parents and my wife

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my supervisor, Frank Dellaert. His knowledge and the unique way of thinking and doing research have been great value for me. Those merits not only greatly influenced my graduate study but also will continue serve me well in the future of my work.

I am also deeply grateful to many researchers who played important roles in my research. They include my thesis committee members, Henrik Christensen, Irfan Essa, Eric Johnson, and Marc pollefeys, who gave me invaluable comments on this work. They also include Anotonio Criminisi, Hailin Jin, and Drew Steedly whom I worked with during my internships and other collaborations. Thank Tucker Balch for suggesting the name of this work. Thank PhotoSynth at Microsoft for supplying the data sets used in the work.

I want to say thank you to my fellow students: Doru-Cristian Balcan, Chris Beall, Alex Cunningham, Alireza Fathi, Pablo Fernandez, Viorela Ila, Yongdian Jian, Michael Kaess, Carlos Nieto, Manohar Paluri, Richard Roberts, Grant Evan Schindler, Duy-Nguyen Ta, and Sang Min Oh. You gave me so much help and made my student life much more enjoyable.

Finally, I must say thank you to my parents and my wife. Without them, it is impossible for me to reach this point. You gave me so much courage and all kinds of support that makes me going forward in all these years. Your love makes nothing impossible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Large-scale mapping has become the key to numerous applications, e.g. Simultaneous localization and mapping (SLAM) for autonomous robots. Despite of the success of many SLAM projects, there are still some challenging scenarios in which most of the current algorithms are not able to deliver an exact solution fast enough. Urban 3D reconstruction is another popular application for large-scale mapping and has received considerable attention recently from the computer vision community. At the heart of urban reconstruction problems is Structure from Motion (SfM). Due to the wide availability of cameras, especially on handhold devices, SfM is becoming a more and more crucial technique to handle a large amount of images.

I will present a novel batch algorithm, namely Tectonic Smoothing and Mapping (TSAM). I will show that the original SLAM graph can be recursively partitioned into multiple-level submaps using the nested dissection algorithm, which leads to the cluster tree, a powerful graph representation. By employing the nested dissection algorithm, the algorithm greatly minimizes the dependencies between two subtrees, and the optimization of the original graph can be done using a bottom-up inference along the corresponding cluster tree. To speed up the computation, a base node is introduced for each submap and is used to represent the rigid transformation of the submap in the global coordinate frame. As a result, the optimization moves the base nodes rather than the actual submap variables. I will also show that TSAM can be successfully applied to the SfM problem as well, in which a hypergraph representation is employed to capture the pairwise constraints between cameras. The hierarchical partitioning based on the hypergraph not only yields a cluster tree as in the SLAM problem but also forces resulting submaps to be nonsingular. I will demonstrate the TSAM algorithm using various simulation and real-world data sets.

Chapter I

INTRODUCTION

Large-scale mapping has become the key enabling technique to numerous applications, e.g. *simultaneous localization and mapping* (SLAM) for autonomous robots [21, 6]. For many scenarios where GPS is not available, e.g., exploring underwater or on a remote planet, how to produce accurate 2D or 3D maps as well as localizing robots' own locations using large-scale mapping and SLAM techniques becomes increasingly important.

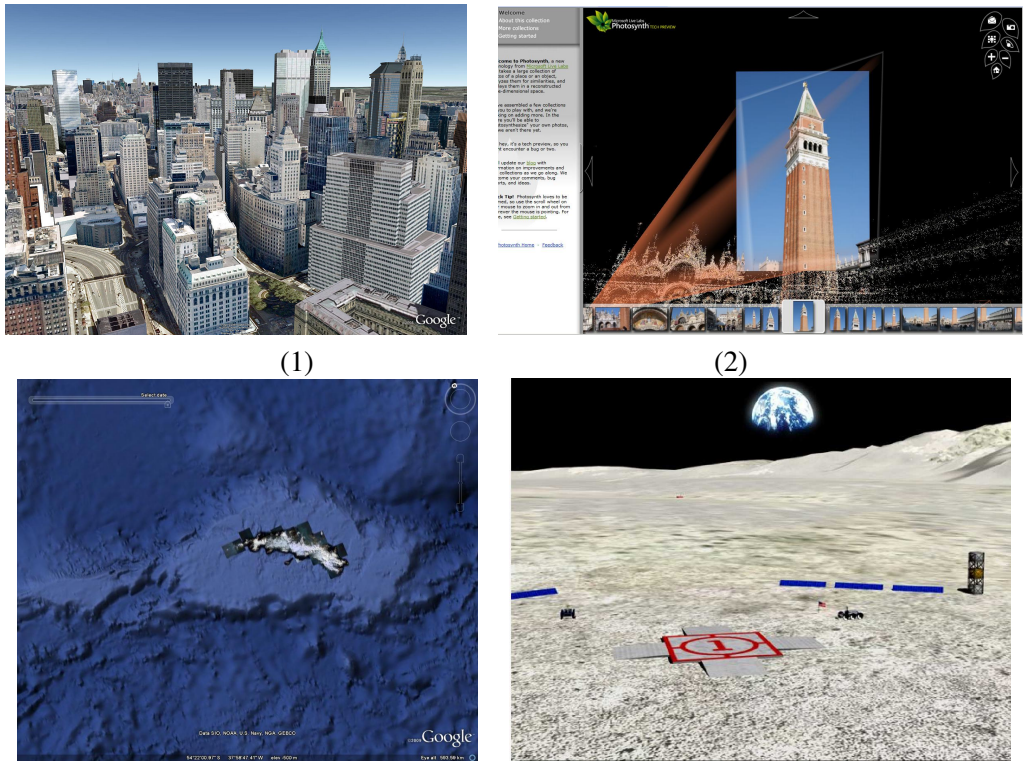
Urban 3D reconstruction is another popular application for large-scale mapping and has received considerable attention recently from the computer vision community. High-quality 3D models are useful in various successful cartographic and architectural applications, such as Google Earth or Microsoft PhotoSynth, as shown in Figure 1. At the heart of urban reconstruction problems is *structure from motion* (SfM) [96]. Due to the wide availability of cameras, especially on handhold devices, SfM is becoming an increasingly indispensable technique to handle the extremely large amounts of images taken every day by people around the world.

1.1 Thesis Statement

My thesis statement is as follows:

> *Tectonic Smoothing and Mapping (TSAM) provides a novel batch algorithm for solving large-scale simultaneous localization and mapping (SLAM) problems and structure from motion (SfM) problems in a divide-and-conquer way, which is fast, exact, robust, and scalable.*

In the remaining part of this chapter, I will lay down the reasoning behind the thesis.



(1) (2)

(3) (4)

Figure 1: **Large-scale mapping has become the key technique to numerous applications.** *(1). The 3D New York on Google Earth; (2) The Microsoft PhotoSynth; (3) The underwater terrain on Google Earth; (4) The simulation of lunar landing.*

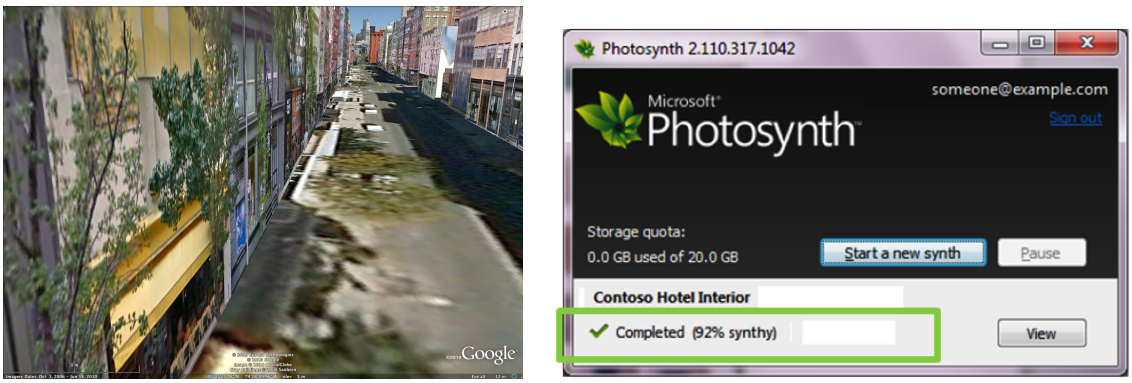


Figure 2: **The complaint on the current 3D mapping products.** *Left: "Nearly all low quality photos . . . no updates have happened in the last 4 years . . . most buildings were very square with flag roofs." – Google Earth Blog; Right: "It is not very funny to wait 120 minutes to realize that your synth sucks." – Wingman2@getsatisfaction.com*

1.2 Motivation

Despite the wide applications of SLAM and SfM techniques, there is much room for improvement over the current user experience, as people desire better services (Figure 2). The complaints about those products are mainly on poor 3D model quality and long waiting time. To speed up the computation, it is very common to use simplified geometric models, such as large squares, to model complex surfaces in the urban reconstruction [84]. An alternative way is to model the 3D point clouds [90, 89, 2] and apply dense stereo algorithm [83, 36], however this approach is considerably more expensive or needs special vehicles and equipments for data collection, which makes it difficult to keep the reconstruction up-to-date.

The *efficiency* is the most important motivation of the proposed work. A fast large-scale mapping algorithm makes more advanced processing possible and also results in more attractive products. In addition to efficiency, *exactness* and *robustness* are also desired in many applications, as inaccurate models or failed mapping may greatly hurt the performance of numerous online applications such as autonomous rescuing.

Broad applicability and *scalability* are the other two goals of this work. Large-scale mapping has become a key technology to many related applications, including the SLAM

problem and the SfM problem. A general algorithm capable of solving both problems will greatly benefit different communities. Moreover, as the sizes of interesting data sets have increased considerably, an algorithm that scales well and is able to process large-scale problems in a parallel and out-of-core manner is also very important.

1.3 Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM) [92] refers to the problem in which a robot manages to build up a map of the environment while keeping tracking of its own location. The SLAM problem has been regarded as a "holy grail" [21, 6] in the robotics community, because of its solution being the key to numerous applications for autonomous robots.

1.3.1 Incremental versus Batch

In the last two decades, a lot of progress has been made in solving the SLAM and SfM problems, and most of the existing approaches can be grouped into two classes. The first class consists of the *incremental* approaches, which include EKF-base SLAM [5, 61, 49], information filters [25, 93], particle filtering SLAM [71, 72], Graphical SLAM[29], TreeMap [32], iSAM [50], etc. As the class name suggests, those approaches gradually build up maps at the same time as the measurements come, hence they are more suitable for the on-line applications. Another advantage of incremental approaches is that it alleviates the challenging initialization problem by initializing a small number of variables each time. In other words, the state variables, i.e. the sensors and the landmarks, are not required to be recovered at once, hence they can typically be computed in a more robust way, such that the estimate is much closer to the global minimum than that generated with a single initialization.

The second class is comprised of *batch* approaches, such as Smoothing and Mapping [19], TORO [44, 41], HOG-Man [40] etc. Those approaches take into account all of the available measurements and generate the estimate of all sensors and landmarks at once. Because all the information is available beforehand, batch approaches have the advantage of

being able to produce a more optimal strategy than the alternative incremental approaches. As a result, batch approaches usually have more potential to achieve a faster and exact solution. In this thesis, we mainly focus on introducing a new batch algorithm that allows the system to fully exploit all the available information.

1.3.2 Challenges

Despite of the recent success of many SLAM projects, there are still some challenging scenarios in which most of the current algorithms are not able to deliver an *exact* solution *fast* enough. One of the challenges is the data-set size, which has increased by several magnitudes over the last decade. A lot of work has been done to push its limits, e.g. [35, 60], and our motivation is to push the envelope even further. The performance also suffers when the data-set size exceeds the size of the system memory, which would obviously make an out-of-core approach appealing.

The second challenge for many real problems is the large amount of noise that is apparent in the measurements, which often yields poor initializations and slows or even leads to failure of the optimization [49]. Due to the nonlinear nature of most SLAM problems, good initializations are necessary for the nonlinear optimization algorithms to converge to the exact solution. Improper initializations not only result in inefficiency of the underlying iterative methods, but also are very likely to cause failure of the entire optimization due to not converging to the desired local minimum. TORO [42] and HOG-Man [40] alleviate the initialization problem by employing stochastic gradient descent algorithm and iteratively optimizing a subset of variables using one measurement each time. On the other hand, SAM [19] uses the Gauss-Newton or Levenberg-Marquardt algorithms to optimize over all variables at once. Compared to the stochastic gradient descent algorithm, the global optimization algorithm requires a better initialization, but typically converges much faster. Attracted by the faster convergence property, we will focus on a novel global optimization algorithm in which the initialization problem can be solved robustly.

1.3.3 Divide and Conquer

Divide-and-conquer is one possible direction towards solving challenging SLAM problems, especially in large-scale environments. The scheme is also referred to as a *submap* based approach [62, 97, 8, 24, 81], as the original map is divided into multiple submaps such that the size of each submap is constrained by some thresholds. In this way, the complexity of optimizing individual submap is bounded.

The divide-and-conquer scheme has been well studied for various optimization problems, especially in the SLAM community. As early as 1976, Brown [9] first employed the submap scheme in the aerotriangulation and mapping of city-scale areas. A recursive partitioning is used to exploit the band diagonal structure of the linear system in the project, and no nonlinearity is considered. The submap idea for SLAM problems was also investigated in hierarchical SLAM [24] with a filtering-based local map building. However, their approach is carried out in two levels, and the map joining at the global level tends to become increasingly expensive with large maps. Later, Paz et al. [81] improved the algorithm by fusing local maps in a hierarchical way, but the overall submap creation scheme was still suboptimal.

The divide-and-conquer approach is closely related to graph-based SLAM algorithms [80, 28, 8, 32, 40], especially tree-based representations have recently gained in popularity but still lack in some aspects. Paskin [79] first employed a tree-based data structure, a so called junction tree, to capture the belief state of robot poses and landmarks in SLAM problems. The algorithm is referred to as a thin junction tree filter (TJTF). To speed up the inference, the tree gets "thinned" periodically by variable contraction, which is proven to minimize the KL divergence before or after the edge removal. Frese [35, 33] introduced another fast algorithm, called treemap, as a back-end for solving large-scale SLAM problems. In addition, treemap applies a sophisticated hierarchical tree partitioning (HTP) to re-balance the binary tree and reduce the cost of propagating the information from leaves to the root. Note that HTP does not produce a junction tree during run-time, and hence

its optimization is not necessarily most efficient. Both TJTF and treemap marginalize the nodes to keep the tree sparse, such that the algorithms become inexact for the same reason as the other filtering approaches. We argue that the marginalization should be avoided for better accuracy.

In both graph theory and linear algebra literature [38, 7], it has been shown that the efficiency of linear system solving depends heavily on the elimination ordering. More recently, the same idea was also applied to the SLAM problem [19]. A good elimination ordering yields small cliques during graph triangulation and introduces fewer non-zero fill-in during matrix factorizations. Although finding an optimal ordering is NP-complete [98], there are two successful schemes for finding a good ordering: *minimum degree* (its variants include MMD [66] and AMD [4]) and *nested dissection* [37, 64].

For submap based approaches, we found that nested dissection has more appealing properties than minimum degree. Although both schemes generate elimination orderings with comparable qualities, nested dissection tends to perform better on large graphs, and its elimination trees are typically lower and better balanced, which naturally fits our hierarchical partitioning. In addition, for solving SLAM problems with planar graphs [58], nested dissection has proven to be optimal. In the work, we use the state-of-art hybrid ordering [39] that combines the advantages of AMD and nested dissection for small and large graphs respectively.

In general, the divide-and-conquer scheme consists of three steps. In the first step, submaps are created such that the dependencies between submaps are as small as possible. Next, each individual submap is optimized independently of other submaps, in a manner similar to non-submap approaches. At last, all the optimized submaps are joined together in a global optimization step.

One important, but commonly neglected aspect, is how to create the submaps, which turns out to be crucial to the global optimization step. Intuitively, the less overlap exists between submaps, the easier the creation of the joint global map becomes. In a typical

incremental approach, a new submap is created when the size of the current submap or its uncertainty exceeds a certain threshold. While this scheme works well when the robot is exploring, the submap joining becomes less efficient when the robot visits previously seen places. In the common case that the robot visits the same place several times, multiple maps have their own copies of the shared area with possibly significantly deviating estimations. It is not only inefficient, but also difficult to compute a consistent solution due to the local minimums. In this work, we argue that, as a batch algorithm has a different perspective at the global level, it produces a more optimal set of submaps and results in a more efficient and robust approach.

Compared to previous incremental approaches in the same divide-and-conquer spectrum, we argue that a batch algorithm enables us to achieve better submap partitioning, which means less overlap between the submaps. From a global perspective, the SLAM graph can be used to supply much more information to the problem decomposition process than local heuristics. A better partitioning in turn results in more efficient and robust approach compared to the traditional approaches.

Our work also shares the same hierarchical idea as HOG-Man [40], which creates hierarchical maps using a distance-based threshold, hence does not necessarily decouple the original problem optimally. Moreover, virtual edges are created between the representation nodes, and the changes are explicitly propagated between successive levels. This scheme not only requires additional computation, but also increases the level of approximation. We are motivated for a more computationally efficient and straight-forward approach in this thesis.

1.4 Structure from Motion

Structure from Motion (SfM) [96, 46, 17] refers to the problem of inferring the structure of the scene and the motion of the camera by using the correspondences between features from different views. Although different sensors are used in the SLAM problem and the

SfM problem, both problems still share a lot of common properties due to the nonlinear optimization nature at the core.

1.4.1 3D Reconstruction and SfM

3D reconstruction was first studied in photogrammetry to build 3D city models from aerial images[9]. Fradkin [30] used stereo reconstruction from aerial images to compute a disparity map and an elevation map under the assumption that the surfaces are planar. Google Earth and Microsoft Live Local also rely on aerial imagery. These systems typically suffer from bad texture quality on the sides of buildings because of the extreme viewing angles.

Recently 3D reconstruction has also been made from ground-level images, as more accurate and better textured models can be created by using ground-level images than by using aerial images. With ground-level imagery, the number of images needed to cover an area is significantly higher [90, 63, 2]. This scheme results in a more challenging 3D reconstruction problem.

At the heart of 3D reconstruction problems is Structure from Motion [96], in which we infer the structure of the scene and the motion of the camera by using the correspondences between features from different views. In particular, certain types of features (points, lines, and so forth) are first extracted and matched across all the image pairs, as shown in Figure 3. Then the camera parameters and feature locations are optimized to minimize a cost function, such as the 2D projection errors, as shown in Figure 4. Tomasi and Kanade first introduced a factorization method for solving the SfM problem [94]. To achieve an Euclidean reconstruction, the non-linear minimization of the projection errors is referred to as *bundle adjustment* in the literature [95].

SfM is becoming more and more useful in many real-life applications, due to the wide availability of cameras, especially on handhold devices. A lot of work [94, 85, 82, 18, 22, 73] has been done to push the SfM technique further. For instance, Davison, et. al. [18, 22, 73] developed monocular SLAM systems for real-time SfM, which mainly focuses

Figure 3: **The data association stage when solving the Structure from Motion problem.** *SIFT features [68] (shown in purple) are first detected from both images and are then matched across the images by posing fundamental matrix constraints. The red lines show the feature flows between the matched feature pairs.*

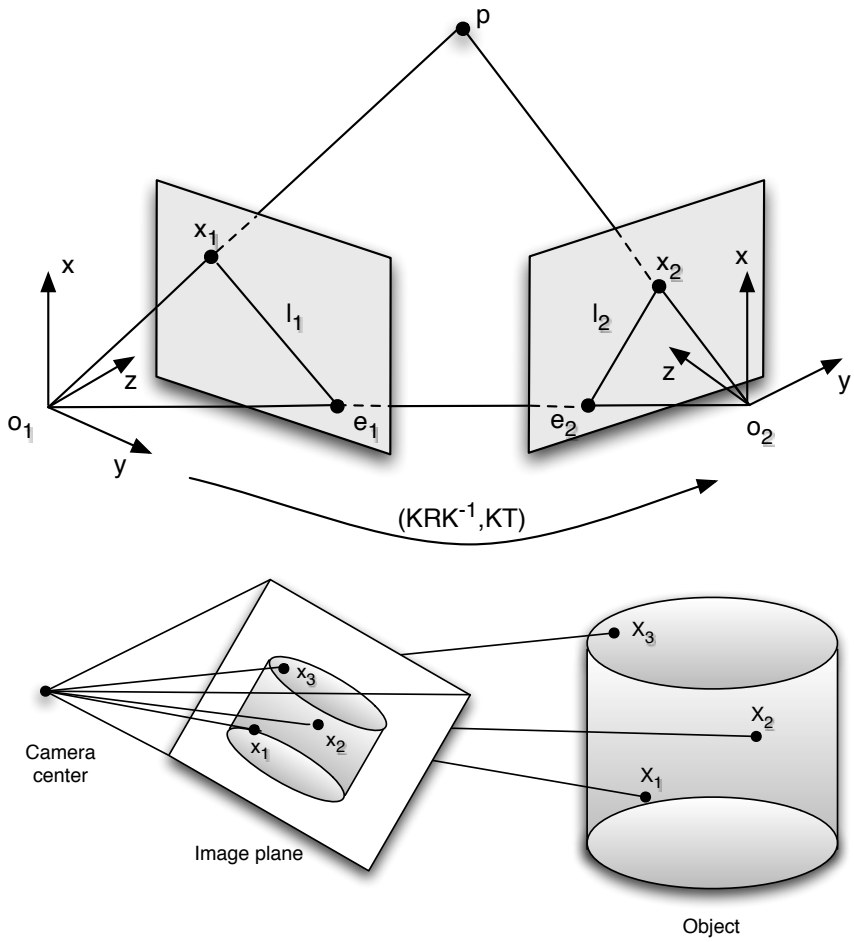


Figure 4: **The geometric reconstruction stage when solving the Structure from Motion problem.** *Left: The fundamental matrix can be computed from the 2D feature correspondences between two images. Right: In the camera resectioning, the camera projection matrix can be computed from the correspondences between 2D features to previously recovered 3D points.*

on online performance, but the scale of the problem is rather limited. In [82], the structure and the motion are first computed from the multi-view relations and then refined using bundle adjustment as the last step. Brown [10] employed an incremental bundle adjustment algorithm to do 3D object reconstruction. In particular, the approach incrementally inserts new frames into the optimization problem, which computes well conditioned initial reconstructions. These experiments mainly focused on relatively small-scale objects and scenes.

1.4.2 Bundle Adjustment

The nonlinear optimization step, also called *bundle adjustment*[95], is a core module in SfM, in which the 3D points and the cameras are jointly optimized to minimize the predefined cost. In many situations it is not practical (or possible) to augment the capture setup in order to avoid the global optimization (bundle adjustment in the SfM literature). Therefore, there has been much work directed at making global optimization more efficient. In the global optimization, it is important to take advantage of the block sparsity structure of the system of equations. In [23], the block-diagonal structure of the Hessian matrix was exploited and the Schur complement was used to first factor out the structure parameters, compute the camera poses, and then back substitute for the structure parameters. For small numbers of cameras, [23] showed that a dense representation for the reduced camera matrix was sufficient. As the number of images increases, the size of the reduced camera matrix increases, and its factorization becomes a bottleneck. At that point, it is necessary to take full advantage of all the sparsity in the system of equations.

There are two main ways to solve a sparse system of equations, iterative approaches such as conjugate gradient, and direct sparse solvers [95]. One advantage of conjugate gradient is that the full Hessian does not need to be stored, substantially lowering the amount of memory used at the expense of computing the error and derivatives many more times. Conjugate gradient methods tend to be competitive with direct linear solvers such as Cholesky

decomposition only when sophisticated preconditioners are used. Jeong [47] presented a carefully tuned system that employs the conjugate gradient algorithm to speed up bundle adjustment. [11] also introduced a multi-scale preconditioning algorithm and applied conjugate gradient to solve the problem. Our approach maintains the computational efficiency of direct solvers while not requiring that the entire Hessian be stored in physical memory at the same time.

There are also many techniques that have been used in large-scale urban reconstruction to avoid having to do a full global optimization. One approach is to augment the image capture system with additional sensors, such as GPS receivers, so that accurate reconstructions can be generated with only local bundle adjustment. Chou [12] used a multi-image triangulation process to build up the feature correspondences and extract the information of lines and surfaces from the urban environment. Akbarzadeh et al. [3] introduced a video-based urban 3D reconstruction system in which the scene structure was computed using the five-point algorithm as described in [77]. However, both approach [3] and [12] heavily rely on accurate camera pose information which is often unavailable in more general systems. Teller developed an urban reconstruction system [91] in which rotations and translations of cameras are decoupled and estimated separately. This approach assumes that extrinsic poses are approximately known, and bundle adjustment is employed to align the rotations of all cameras. In addition, the system requires that images in the same set share the same optical center and that the scene contains enough line features.

1.4.3 Towards Large Scale

Large-scale structure from motion (SfM) problems have gained more and more attention lately, as SfM is becoming one of the key technologies in applications such as city-scale 3D reconstruction. A lot of effort has been made to push SfM algorithms towards collections of a large number of photos [90, 74, 63, 83, 1, 47, 31]. Both skeletal graphs [89] and iconic scene graphs [63] try to capture a compact summary of 3D environments. In this

thesis, we concentrate on the back-end optimization phase, after feature extraction and data association has been performed (Figure 3), which are daunting problems in their own right [90, 83, 31].

In photogrammetry, divide and conquer approaches are a common and popular way to "bundle" data from a large area [9]. When images are taken sequentially from a plane or a ground vehicle, a simple way to generate sub-problems is to make partitions after a fixed number of frames [87]. However, this approach is decidedly suboptimal when there are a lot of "loop closures" in the camera trajectory, which is typically the case in unstructured photo-collections. Moreover, for such unordered, wide-baseline data-sets we typically do not have knowledge of the capture ordering, making this approach unsuitable.

Another important problem worth investigating is how to avoid degeneracies when generating submaps. General partitioning algorithms [51] do not take into account the domain knowledge of SfM problems. Hence, directly applying those algorithms will easily introduce degeneracies to the state variables, especially 3D points, as each 3D point is typically only visible in a small number of cameras (two or three in practice). In fact, little work has been done on how to optimally divide the SfM problem while keeping all the individual sub-problems fully constrained.

Divide-and-conquer methods to efficiently solve large-scale bundle-adjustment problems have long been favored in the photogrammetry community [9, 95]. In structure from motion problems, which are typically much more unstructured, dividing the entire problem into small pieces not only makes the bundle adjustment optimization more scalable, but also provides an easier way to generate good initializations, as has already been demonstrated to be a crucial capability for SfM problems [23].

The divide-and-conquer idea for continuous optimization was first explored in the linear algebra community under the name "nested dissection" [37]. Lipton, Rose, and Tarjan subsequently showed that, for certain classes of graphs (e.g., planar), separator theorems

exist that enable one to obtain theoretical guarantees on the worst case computational complexity [64]. In detail, the nested dissection (ND) algorithm recursively partitions the graph of the original problem and finds a *vertex separator* V_S which splits the graph into two parts A and B, such that there are no connections between any node in A and any node in B. More recently, Krauthausen et. al. [58] applied the separator theorems of the nested dissection algorithm to large-scale urban mapping problems in robotics.

A graph-based approach has also been investigated in SfM. Ni et.~al.~[74] introduced an out-of-core SfM algorithm which uses submaps to leverage a computational advantage. However, in their approach, only a single-level submap structure is employed, and the main bottleneck is solving the dense linear system corresponding to a large separator. Moreover, their approach does not handle degeneracies in the submaps explicitly. In this thesis, we introduce a recursive partitioning approach to produce submaps of very small size, and also address the degeneracy problem by working on a hypergraph representation.

1.5 Tectonic Smoothing and Mapping

In this thesis, we introduce a novel *multi-level* batch SLAM algorithm, namely TSAM, that employs the *nested dissection* algorithm [37, 64] to solve SLAM problems in an efficient, robust, and *exact* manner. After recursively partitioning the original SLAM graphs (Figure 5), we can represent the decoupled SLAM problem by a *cluster tree.* As defined in [57], a cluster tree is a directed tree of clusters in which the running intersection property holds, and each factor in the original graph is associated with a cluster. In fact, the cluster tree is more general than the junction tree or the clique tree [7], which have already been widely used in the graphical model based inference. Here we simply use the cluster tree to organize our SLAM computation.

Previously we introduced a preliminary version of the current algorithm in [75], which was a two-level submap based approach. However, the algorithm does not have the ability to maintain hierarchical maps, and hence does not scale well enough. In addition, it uses

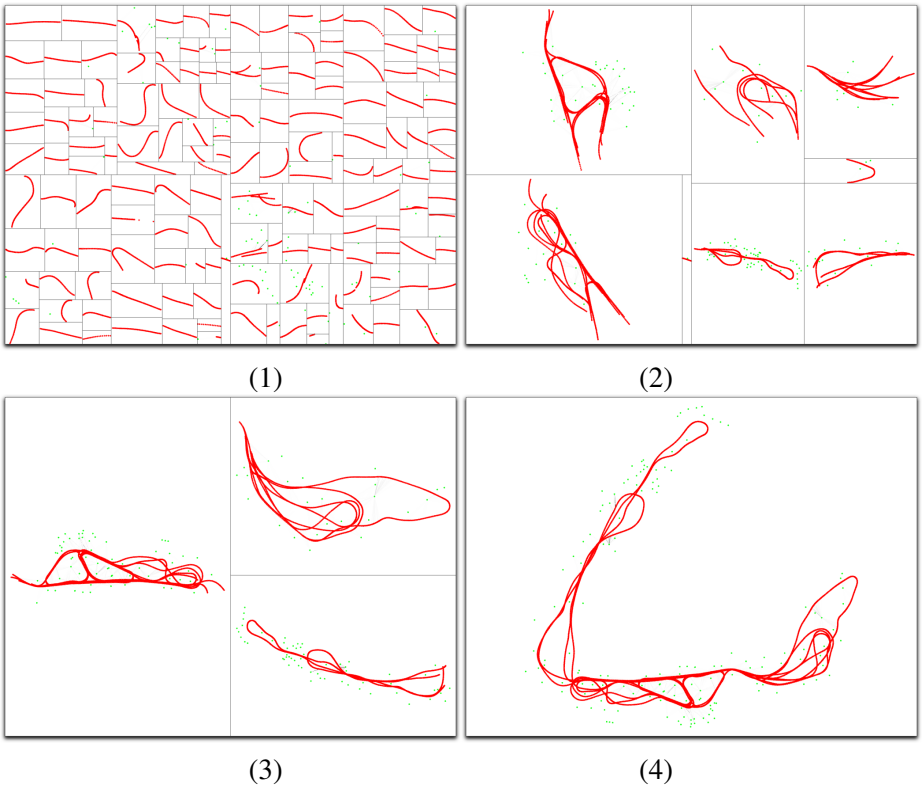


Figure 5: **TSAM recursively partitions the SLAM graph into a submap tree, and the optimization proceeds from the leaves to the root.** *Following the treemap visualization [86], each rectangle represents a submap, and the sub-rectangles represent the submaps in the child level. The red and green dots are robot poses and landmarks respectively. 1).the finest level of submaps; 2).the second coarsest level of submaps; 3).the coarsest level of submaps; 4).the optimized full map.*

edge separators to partition the SLAM graph and generates fully connected separators, which slows down the computation for large maps. The algorithm also requires multiple iterations between the submaps and the separator to converge to exact minima. Our motivation is to address these issues in the new algorithm we are going to introduce. More recently, Kim et al. [55] integrated TSAM into a multi-robot setting together with iSAM [50], and each robot maintains its own submap. The resulting system is able to merge and update the maps from multiple robots in real-time, but all the maps have to be merged at once as the result of one-level submaps.

We also introduce *base nodes* to speed up the convergence of nonlinear optimization, and fully exploit the power of the submap representation. The intuition here is rather straight-forward: if individual submaps have been optimized properly, they only need to move by some unknown rigid transformation with respect to each other in the global optimization step. Such an effect can be achieved by introducing a base node for each submap. Instead of optimizing over all the variables as in the traditional approaches, we move all the variables in the submaps as bundles represented by their base nodes, hence our algorithm is able to work on a much smaller set of unknown variables and run faster.

1.6 Dissertation Overview

The contributions of this thesis are as follows:

1. I introduce a principled way of applying the nested-dissection style divide-and-conquer scheme to the SLAM problem and the SfM problem. The detail will be covered in Chapter 2.

2. TSAM has four appealing properties (discussed in Chapter 3), which are

   (a) *TSAM is fast*: The underlying graph structure is exploited using the nested dissection algorithm, and the hierarchical submap based scheme greatly enhances the efficiency. To reduce the number of nonlinear iterations, TSAM uses base

nodes to reuse the previously optimized submaps and achieve fast convergence.

(b) *TSAM is exact*: The TSAM algorithm is derived from Smoothing and Mapping (SAM) [19], and no approximations are made during the optimization.

(c) *TSAM is robust*: The estimate is incrementally computed by solving sub-problems and is gradually extended to the entire problem, which alleviates the initialization problem and makes the algorithm much more robust.

(d) *TSAM is scalable*: The partitioned submaps can be independently solved and later merged by integrating the corresponding separators, hence TSAM is capable of solving the SLAM/SfM problems in an out-of-core manner.

3. I introduce a constraint-aware partitioning method for the SfM problem using hypergraphs in Chapter 4, which guarantees that all the sub-problems are fully constrained.

The in-depth discussion will be presented in Chapter 5.

Chapter II

TECTONIC SMOOTHING AND MAPPING

In this chapter, we introduce the main part of the TSAM algorithm. First we introduce the formulation of SLAM and SfM problems as well as their factor graph representation. Second we explain the TSAM algorithm in two parts: the divide step and the conquer step. Due to the nonlinear nature of SLAM and SfM problems, we also further split the the conquer step into the linear case and the nonlinear case, both of which use the same partitioned graphs from the divide step.

2.1 Problem Formulation

In this section, we use an exemplar SLAM problem (Figure 6) to show that the SLAM problem can be encapsulated by a factor graph representation [19] as well as the corresponding numerical counterpart. The mixture of both graph and matrix interpretation lends to a unique perspective of solving the SLAM problem.

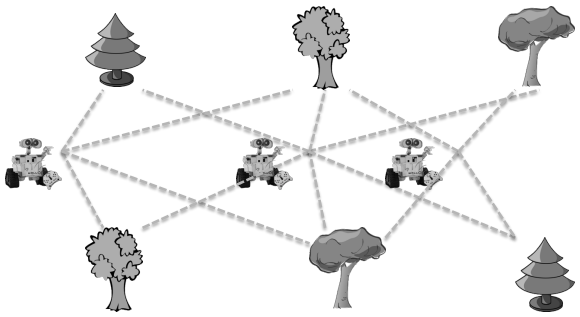


Figure 6: **An exemplar SLAM problem.** *The robot moves for three steps, and it sees some or all of the six landmarks in each step.*

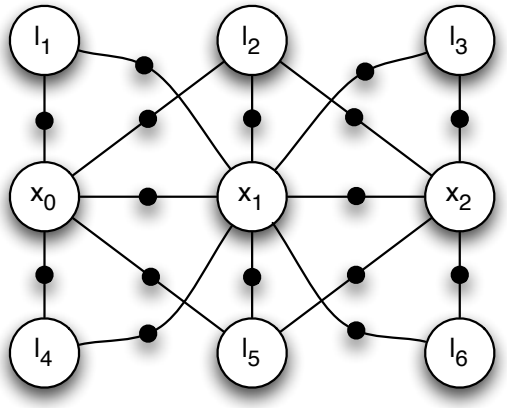


Figure 7: **The factor graph of an exemplar SLAM problem.** *As a bipartite graph, there are two types of nodes: the variable nodes as circles and the factor nodes as black dots.*

2.1.1 SLAM and SfM as A Factor Graph

We use the same formulation as used by Dellaert and Kaess in [19], and denote the robot poses as $X = \{x_i\}$ with $i \in [0, M]$ and the landmarks as $L = \{l_j\}$ with $j \in [1, N]$. The joint probability of the robot poses and the map can be then formulated as:

$$P(X, L, Z) \propto \prod_{i=1}^{M} P(x_i | x_{i-1}, u_i) \prod_{k=1}^{K} P(z_k | x_{i_k}, l_{j_k}) \tag{1}$$

where u_i is the control input at step i, and z_k is the measurement of landmark l_{j_k} at robot pose x_{i_k}.

We employ *factor graphs* [59] to represent the SLAM problem. As illustrated in Figure 7, the factor graph G is a bipartite graph and can be denoted as a tuple $(\mathscr{F}, \Theta, \mathscr{E})$, where $\mathscr{F}$ is representative of the factor nodes corresponding to the constraints (odometry or landmark measurements), Θ denotes the variable nodes corresponding to unknowns X and L, and $\mathscr{E}$ represents the edges connecting $\mathscr{F}$ and Θ. Indeed, the factor nodes define the joint probabilities over their involving variables in Equation 1, and the factor graph G defines its factorization:

$$P(X, L, Z) \propto f(G) = \prod_{i} f_i(\Theta_i) \tag{2}$$

where Θ_i is the set of variables connected to factor f_i. One way to carry out inference on a

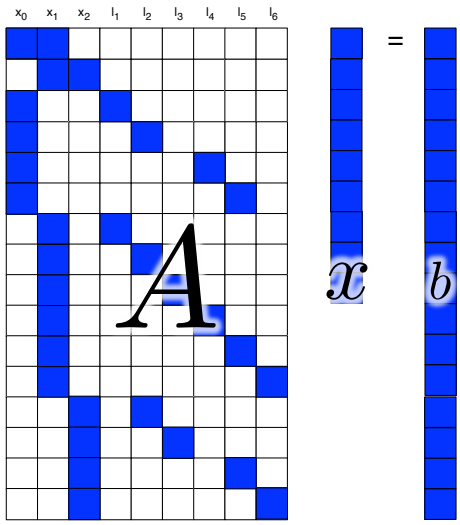


Figure 8: **The matrix perspective of the exemplar SLAM problem.** *The original non-linear problem is converted to a linear least-square problem. The Jacobian matrices F, G, H, and J are assembled to formulate A, and the error terms a and c are assembled to formulate the right-hand-side b.*

factor graph is Smoothing and Mapping [19], in which all the variables Θ are eliminated in a specific ordering, and the resulting Bayes net can be solved in a back-substitution phase.

2.1.2 SLAM and SfM as Least-Square Problems

Furthermore, we assume Gaussian noise in measurement models as is standard in SLAM literature, and the SLAM problem can be converted to a least squares problem [21]:

$$\begin{aligned} -\log\left(f(\Theta)\right) \quad &\propto \quad \sum_i \left\| q_i(x_{i-1}, u_i) - x_i \right\|^2_{\Lambda_i} \\ &+ \quad \sum_k \left\| h_k(x_{i_k}, l_{j_k}) - z_k \right\|^2_{\Sigma_k} \end{aligned} \tag{3}$$

where q_i is the motion model, and h_k is the measurement model. Both the models have zero-mean Gaussian noise with covariance matrices Λ_i and Σ_k respectively. Here $\|e\|^2_\Sigma \triangleq e^T \Sigma^{-1} e$ denotes the squared Mahalanobis distance given the covariance matrix Σ.

The maximum a posteriori (MAP) estimate Θ^* of the robot poses X and the landmarks L can be obtained by maximizing the joint probability $P(X, L, Z)$, which is equivalent to

minimizing the negative log-likelihood in Equation 3:

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \ -\log(f(\Theta)) \tag{4}$$

By linearizing the nonlinear functions q_i and h_k at the latest estimate, we convert the nonlinear least-square problem in Equation 3 to a linear least-square problem:

$$\delta^* = \underset{\delta}{\operatorname{argmin}} \ \left\{ \sum_{i=1}^{M} \left\| F_i^{i-1} \delta x_{i-1} + G_i^i \delta x_i - a_i \right\|_{\Lambda_i}^2 + \sum_{k=1}^{K} \left\| H_k^{i_k} \delta x_{i_k} + J_k^{j_k} \delta l_{j_k} - c_k \right\|_{\Sigma_k}^2 \right\} \tag{5}$$

in which F_i^{i-1} and G_i^i are the Jacobians of $q_i(\cdot)$ with respect to x_{i-1} and x_i. $H_k^{i_k}$ and $J_k^{j_k}$ are the Jacobians of $h_k(\cdot)$ with respect to x_{i_k} and l_{j_k}.

By pre-multiplying the Jacobian matrix in Equation 5 with $\Lambda_i^{-T/2}$ and $\Sigma_k^{-T/2}$, we may rewrite the cost function in the format of a standard least-square problem:

$$\delta^* = \underset{\delta}{\operatorname{argmin}} \ \|A\delta - b\|_2^2 \tag{6}$$

where A is obtained by collecting all the Jacobian matrices F_i^{i-1} , G_i^i, $H_k^{i_k}$, and $J_k^{j_k}$. b is the corresponding right-hand side (RHS).

Solving the least-squares problem as in Equation 6 can be done by QR factorization, which is exactly equivalent to variable elimination in graphical models, as described in Section 2.1. We refer interested readers to [19] for more details. In this work, we essentially introduce a new algorithm to solve the optimization problem in Equation 6 in a more efficient and robust manner.

2.2 Divide Step

In the section, we explore how the original SLAM problem can be divided into a set of sub-problems. In particular, we show that it is possible for a batch algorithm to exploit the graph structure of the SLAM problem using the nested dissection algorithm [64], and obtain a good partitioning that makes the dependencies between the sub-problems small.

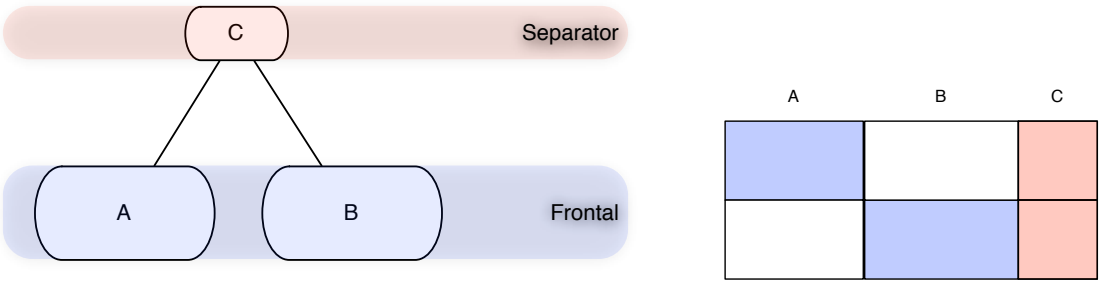


Figure 9: **The separator C and the frontal variables $\{A, B\}$ generated by nested dissection algorithm.** *Left: the frontal variables A and B are statistically independent given their separator C. Right: the columns of the corresponding matrix are reordered using the nested dissection ordering.*

2.2.1 Nested Dissection

Nested dissection was first introduced by George [37] to solve linear systems defined on square grids, and the algorithm was later generalized by Lipton [64] for any linear system defined on a planar or almost-planar graph. In addition, for solving SLAM problems with planar graphs [58], nested dissection has seen proven to be optimal.

The basic idea behind nested dissection is to recursively find small vertex separators such that at each level the remaining two subgraphs are disconnected. In other words, given a matrix M and its corresponding graph G, nested dissection continuously partitions as follows: each time the current subgraph G_i is split into three sets A_i, B_i, and C_i, such that no vertex in A_i is connected to any vertex in B_i, as illustrated in Figure 9. Note that nested dissection does not guarantee that A_i or B_i is a connected graph. In the disconnected case, we simply make each disconnected component a new subgraph. Hence the tree induced by nested dissection is usually a K-way tree rather than a binary tree. Without loss of generality, we use the nested dissection notations above and assume we always obtain two-way cuts.

The sets A_i, B_i are referred to as the *frontal variables*, and C_i is called the *separator* of A_i and B_i. The nodes in submap A_i and B_i are grouped together and ordered first, the separator nodes in C_i are ordered last. Hence, the columns of the sub-matrix M_i can be

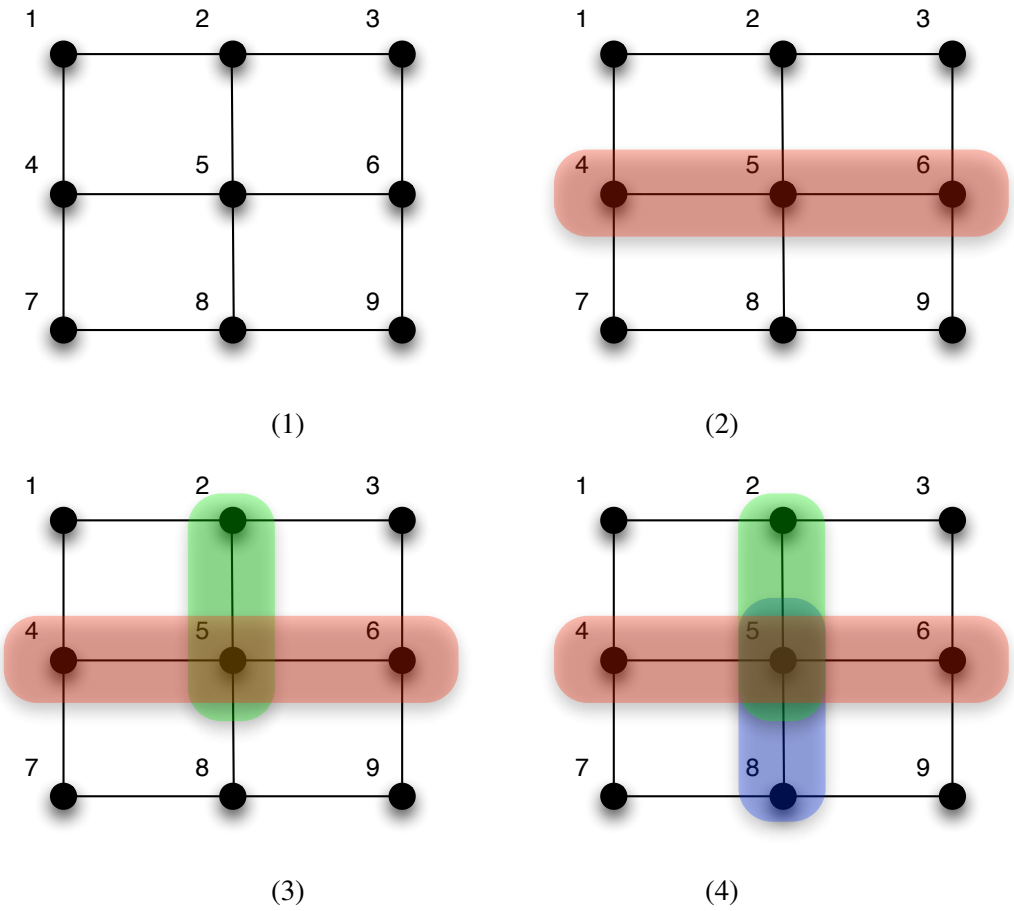


Figure 10: **The nested dissection algorithm is applied to a** 3×3 **mesh.** *(1). The original graph; (2) The separator in red splits the graph into two parts. (3) The green separator splits the upper part. (4) The blue separator splits the bottom part.*

ordered accordingly, as shown in the right of Figure 9.

The partitioning above can be applied recursively to any graph. For example, we may first divide the 3×3 mesh graph in Figure 10 using the separator in the red shadow and then proceed by choosing the separators in the blue and green shadow.

Such a series of partitioning yields an ordering called the *nested dissection ordering*, which is 1, 3, 2, 7, 9, 8, 4, 5, 6 in this example. Based on the obtained ordering, we may perform variable elimination on the graphical model, which corresponds to applying matrix factorization on the matrix with reordered columns, as shown in Figure 11. Because submap variables do not have connections to variables in other submaps, the inference tasks

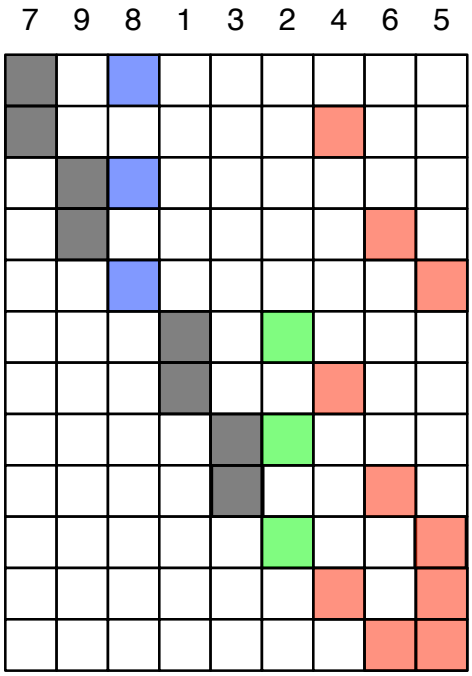


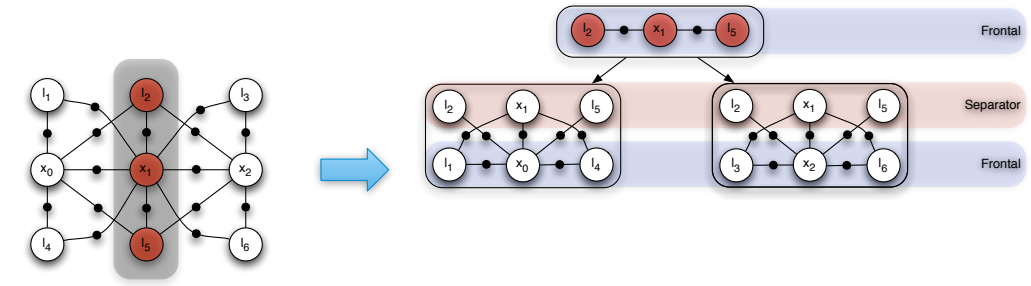
Figure 11: **The nested dissection ordering for the** 3×3 **mesh in Figure 10.**

in individual submaps can be carried out in parallel.

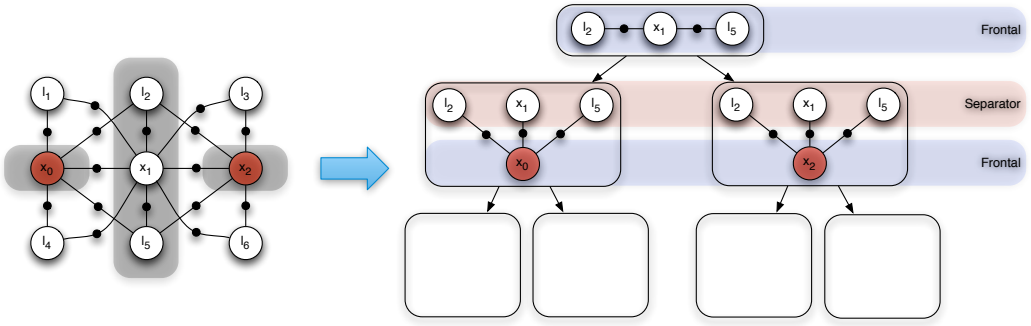
2.2.2 Partition the SLAM Graph

To create a hierarchical set of submaps, we *recursively partition* the SLAM graph using nested dissection. More specifically, we use METIS [53] to find a nested dissection ordering at the global level and then order the resulting subgraphs locally using AMD algorithm [4].

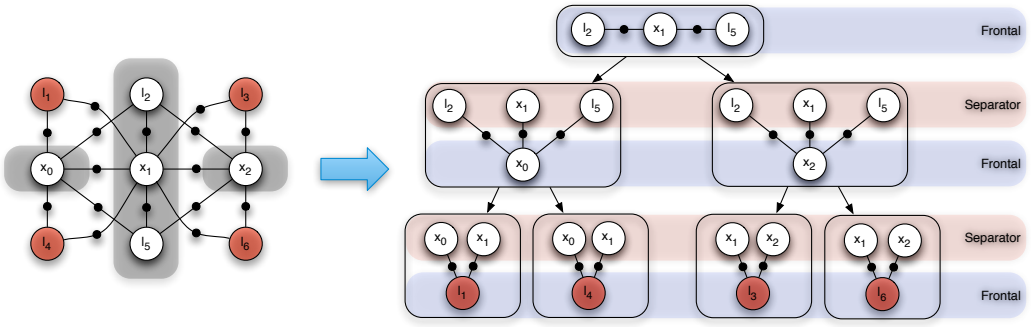
In the context of the SLAM problem represented by a factor graph G, nested dissection distributes all the factors $\mathscr{F}$ along a cluster tree T [57], as shown in Figure 12. The recursive partitioning starts from the root and continuously builds the child subgraphs. For a certain subgraph in the cluster tree, let us assume that we have the remaining factors $\mathscr{F}_0$, the frontal variables Θ_0, and the separator Θ_S inherited from its parent (in the case of the root node, $\mathscr{F}_0 = \mathscr{F}$ and $\Theta_S = \emptyset$). The nested dissection algorithm groups the variables Θ_0 into three sets Θ_A, Θ_B and Θ_C, such that no variable in Θ_A shares any factor with variables in Θ_B.



(1)



(2)



(3)

Figure 12: **The SLAM graph is recursively partitioned using the nested dissection algorithm.** *(1). First l_1, x_1, and l_5 are chosen to split the original SLAM graph and formalize the first submap. The two blank rectangles correspond to the left subgraph and the right subgraph which have not been partitioned yet. (2). x_0 and x_2 are chosen to split the left subgraph and right subgraph respectively, and they also serve as the frontal variables of two second-level submaps. (3) As all the four remaining submaps only have one node, there is no further partitioning, and they are placed as the third-level submaps.*

Based on the frontal variables Θ_C, we define submap $\mathscr{M}$ as a tuple of factors, the frontal variables, and the separator:

$$\mathscr{M} = (\mathscr{F}_C, \Theta_C, \Theta_S) \tag{7}$$

where $\mathscr{F}_C$ are the factors in $\mathscr{F}_0$ not connected to any variable in Θ_A and Θ_B, notated as $E(\mathscr{F}_C, \Theta_A) \cup E(\mathscr{F}_C, \Theta_B) = \emptyset$. The frontal variable set of the new submap $\mathscr{M}$ is Θ_C, and the new separator of $\mathscr{M}$ is Θ_S. The remaining factors can be grouped into two sets $\mathscr{F}_A$ and $\mathscr{F}_B$ with respect to Θ_A and Θ_B, such that $E(\mathscr{F}_A, \Theta_A) \neq \emptyset$ and $E(\mathscr{F}_B, \Theta_B) \neq \emptyset$. We may find the separators of two child nodes as $\Theta_{AC} \subset \Theta_C \cup \Theta_S$ with $E(\mathscr{F}_A, \Theta_{AC}) \neq \emptyset$ and $\Theta_{BC} \subset \Theta_C \cup \Theta_S$ with $E(\mathscr{F}_B, \Theta_{BC}) \neq \emptyset$. Such a factorization can be written as

$$f(G_0) = f(\Theta_A) f(\Theta_B) f(\Theta_C)$$

The recursive partitioning exits when the size of the current subgraph is below a certain threshold α ($\alpha = 40$ in our experiments). As all the factors in the original factor graph G have been distributed to the nodes of cluster tree T, we may format Equation 2 in terms of a cluster tree:

$$f(G) = \prod_{i \in T} \prod_{f_j \in \mathscr{F}_i} f_j(\Theta_{j_k})$$

where Θ_{j_k} are the variables associated with the factor f_j. All the nodes in the subgraphs can be further ordered using AMD. The pseudo-code is listed in Algorithm 1. The collection of all the clusters (submaps) forms the cluster tree.

2.3 Conquer Step: Linear Systems

So far we have introduced how the original problem can be partitioned into smaller sub-problems. The main idea of the conquer stage is typically as follows: first the submaps are optimized locally, and their relative positions and orientations are determined when optimizing their parent submap. The same idea applies to our cluster tree representation as well: the inference is first done locally inside each submap by eliminating frontal variables

Algorithm 1 Recursively partition the input factor graph $G = (\mathscr{F}, \Theta, \mathscr{E})$ and build a cluster tree which encapsulates all the factors. The recursive algorithm starts by calling *root=Partition*($\mathscr{F}$, Θ, $\emptyset$).

Function: $\mathscr{M}$ =***Partition***($\mathscr{F}_0$, Θ_0, Θ_S)
if *size*$(G) > \alpha$ **or** G_0 is not fully connected
  $(\Theta_A, \Theta_B, \Theta_C)$ =***Nested_Dissection***(Θ_0);
  $\mathscr{F}_0 \longrightarrow \mathscr{F}_A \cup \mathscr{F}_B \cup \mathscr{F}_C$;
  find Θ_{AC} and Θ_{BC};
  $\mathscr{M} = (\mathscr{F}_C, \Theta_C, \Theta_S)$;
  $\mathscr{M}$.*left_child*=***Partition***$(\mathscr{F}_A, \Theta_A, \Theta_{AC}))$;
  $\mathscr{M}$.*right_child*=***Partition***$(\mathscr{F}_B, \Theta_B, \Theta_{BC}))$;
else
  $\mathscr{M}$ =$(\mathscr{F}_0, \Theta_0, \Theta_S)$;
end
return $\mathscr{M}$;

and is then finished by a back-substitution step. For linear systems, this technique is also referred to as a multifrontal method [39].

2.3.1 Leaves-to-Root Elimination

First, we apply an elimination algorithm to the partitioned subgraphs obtained from Algorithm 1 in leaf-to-root order. The frontal variables $F_{\mathscr{M}}$ (Θ_C in Equation 7) are eliminated for each submap graph $\mathscr{M}$, which yields a directed subgraph as shown in Figure 13. The pseudo-code is in Algorithm 2.

The elimination is indeed refactorizing the factor graph contained in submap $\mathscr{M}$, and it is equivalent to applying the chain rule to the joint probability of the frontal variables $F_{\mathscr{M}}$ and the separator variables $S_{\mathscr{M}}$ (Θ_S in Equation 7) :

$$P(F_{\mathscr{M}}, S_{\mathscr{M}}) = P(F_{\mathscr{M}}|S_{\mathscr{M}})P(S_{\mathscr{M}}) \tag{8}$$

Note that $P(S_{\mathscr{M}})$ corresponds to the new factors between the separator variables and is propagated to the parent submap, shown as squares in Figure 13. For each cluster $\mathscr{M}$, as $P(F_{\mathscr{M}}|S_{\mathscr{M}})$ is a Gaussian density, eliminating frontal variables is equivalent to factorizing the corresponding matrix:

$$P(F_{\mathscr{M}}|S_{\mathscr{M}}) \propto \exp -\frac{1}{2} \|R_{\mathscr{M}} F_{\mathscr{M}} + A_{\mathscr{M}} S_{\mathscr{M}} - d_{\mathscr{M}}\|^2_{\Sigma_{\mathscr{M}}} \tag{9}$$

Algorithm 2 The elimination of the frontal variables $F_{\mathscr{M}}$ in the submap $\mathscr{M}$ using AMD orderings. The recursive algorithm generates a set of new factors $\mathscr{F}_S$ propagated to the parent cluster of $\mathscr{M}$.

Function: $\mathscr{F}_S$=**EliminateNode**($\mathscr{M}$)
with $\mathscr{M} = (\mathscr{F}_{\mathscr{M}}, F_{\mathscr{M}}, S_{\mathscr{M}})$
if $\mathscr{M}$.**hasChildren**()
  $\mathscr{F}_A$ =**EliminateNode(**$\mathscr{M}$.*left_child*);
  $\mathscr{F}_B$ =**EliminateNode(**$\mathscr{M}$.*right_child*);
  (*clique*, $\mathscr{F}_S$)=**Triangulate**($\mathscr{F}_{\mathscr{M}} \cup \mathscr{F}_A \cup \mathscr{F}_B$);
else
  $\mathscr{F}_S$ =**Triangulate**($\mathscr{F}_{\mathscr{M}}$);
end
return $\mathscr{F}_S$;

in which $\begin{bmatrix} R_{\mathscr{M}} & A_{\mathscr{M}} \end{bmatrix}$ is the factorized matrix with $R_{\mathscr{M}}$ being upper triangular. $d_{\mathscr{M}}$ is the corresponding right-hand side (RHS), and $\Sigma_{\mathscr{M}}$ is the covariance matrix. The matrix factorization is currently done using SuiteSparseQR [14].

2.3.2 Root-to-Leaves Back-Substitution

Once the elimination step is done, the optimal values of all the variables can be obtained by performing back-substitutions in the root-to-leaf order. The process starts from the root cluster and recursively solves the children. For each cluster $\mathscr{M}$ with frontal variables $F_{\mathscr{M}}$, as all the parent clusters have been solved, we may compute frontal variables $F_{\mathscr{M}}$ using the estimate of separator variables $S_{\mathscr{M}}$:

$$F_{\mathscr{M}} = R_{\mathscr{M}}^{-1}(d_{\mathscr{M}} - A_{\mathscr{M}} S_{\mathscr{M}})$$

2.4 Conquer Step: Nonlinear Systems

In this section, we extend the algorithm for linear systems we introduced in the last section to the nonlinear case. Due to the nonlinear nature of the SLAM problems, linearization errors prevent the system from converging within one pass of elimination and back-substitution. Hence, iterative methods are often employed to tackle the linearized system at

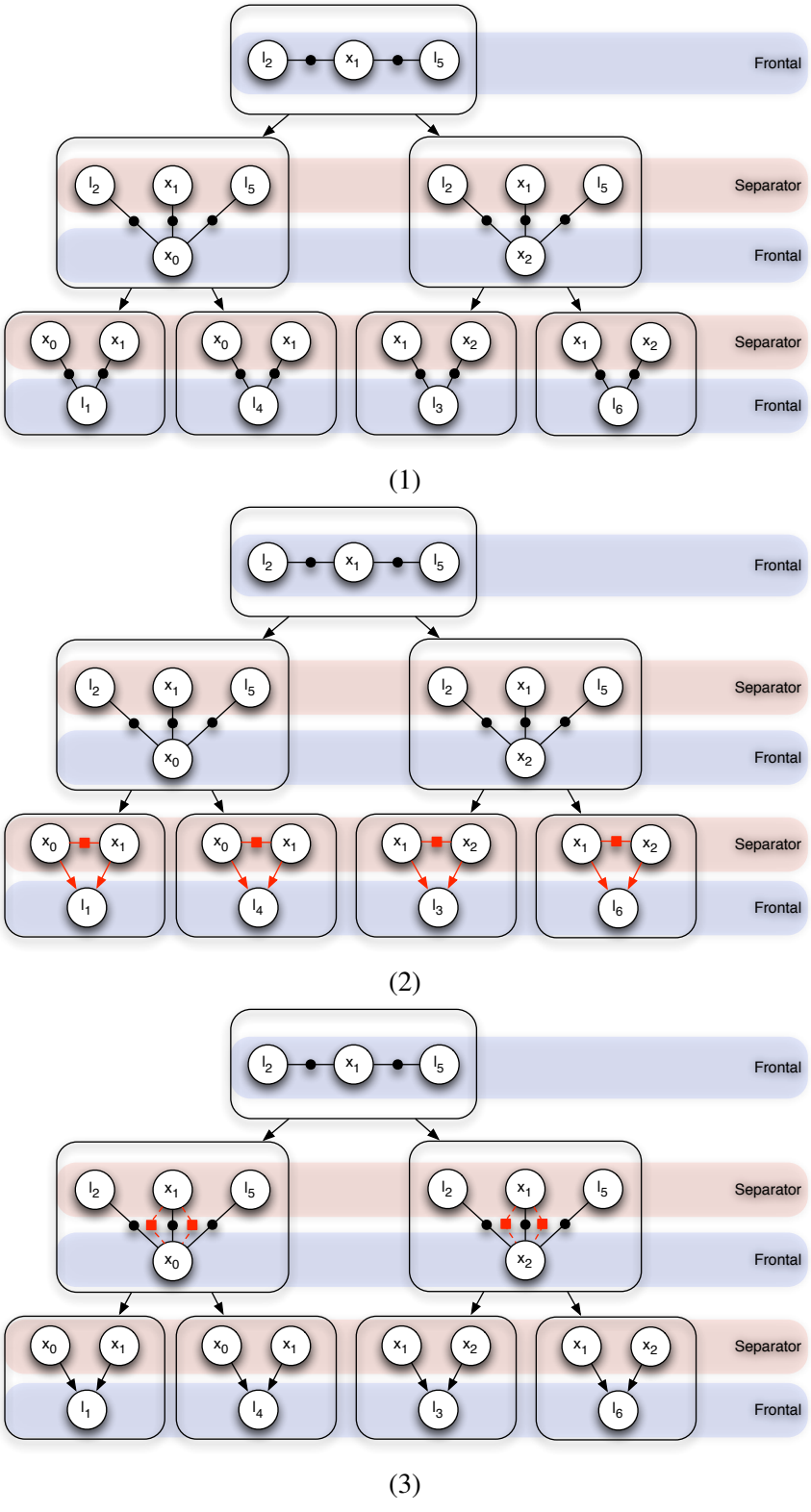
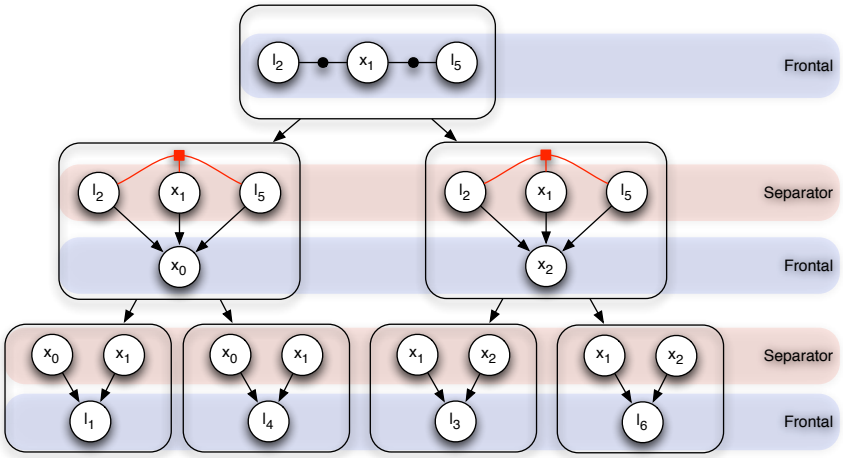
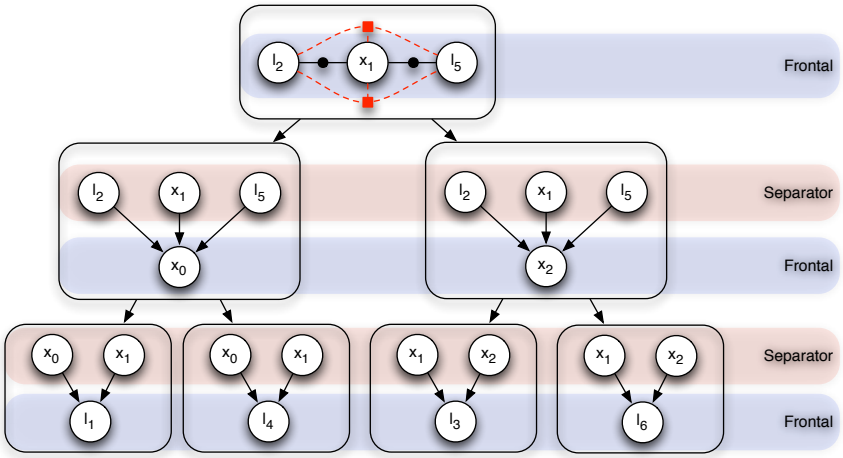


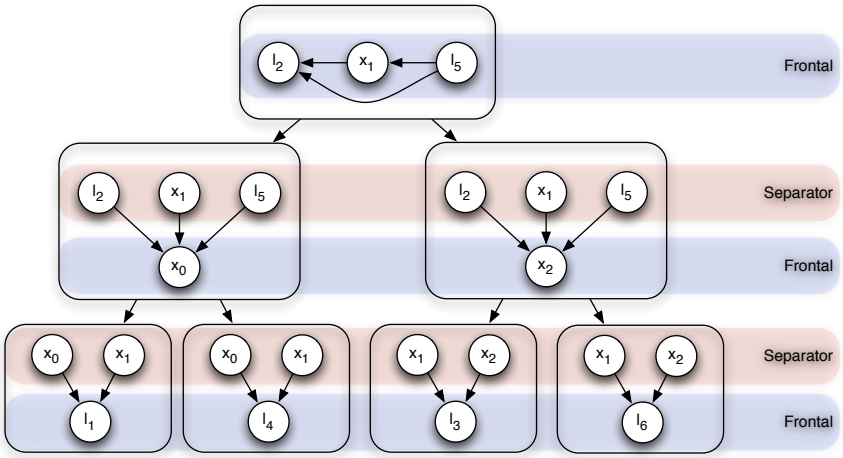
Figure 13: **Variable elimination for the cluster tree created in Figure 12.** *(1) The original cluster tree; (2). The frontal variables of the leaf submaps are eliminated, and four new factors are generated as labelled in red; (3) The four new factors are propagated to the second-level submaps.*



(4)



(5)



(6)

Figure 14: **Continue on variable elimination in Figure 13.** *(4) The frontal variables of the second-level submaps are eliminated, and two new factors are generated as labelled in red; (5) The two new factors are propagated to the root submap. (6) The variable elimination is applied to the root submap.*

the latest estimate. Below we show how such nonlinear optimization adapts to the multiple-level submaps.

2.4.1 Subtree Optimization

In the hierarchical context, the subtree optimization mainly serves two purposes. Assume the subtree with root cluster $\mathscr{M}$ is denoted as $\mathscr{T}_{\mathscr{M}}$, and $\mathscr{T}_{\mathscr{M}} = \mathscr{M} \cup \mathscr{T}_{\mathscr{M}_1} \cup \cdots \cup \mathscr{T}_{\mathscr{M}_s}$ in which s is the number of $\mathscr{M}$'s child clusters. First, subtrees $\left\{\mathscr{T}_{\mathscr{M}_k}\right\}$ are aligned together. Second, the relative positions between the frontal variables $F_{\mathscr{M}}$ of $\mathscr{M}$ are determined. The first task can be done by using the factors propagated from $\left\{\mathscr{T}_{\mathscr{M}_k}\right\}$, and the second task can be achieved by using the factors stored locally in the cluster $\mathscr{M}$. In practice, all those factors constitute a new nonlinear factor graph used for the submap optimization.

One naive way to optimize over the resulting nonlinear factor graph is to optimize over the child variables $C_{\mathscr{M}}$ in $\left\{\mathscr{T}_{\mathscr{M}_k}\right\}$ and $F_{\mathscr{M}}$ together. However, this approach does not exploit the fact that each subtree $\mathscr{T}_{\mathscr{M}_k}$ has usually been well recovered during the previous optimization. If the size of variable set $C_{\mathscr{M}}$ is large, such an optimization process is not only inefficient but also tend to get stuck in local minima. We will introduce how to speed up this step in the next chapter.

2.4.1.1 Subtree Full Optimization

Once the subtree alignment is finished, we may invoke a full optimization step to further balance all the variables in the subtree whose root is the current submap. An example of this process is illustrated in Figure 15. In this step, all the base nodes stay fixed, and the resulting optimization is equivalent to the traditional SAM [19] approach. As both the local structure of child submaps and the current submap are well optimized, we found at most two iterations of the full optimization suffices for all clusters except the root. For the root cluster in the cluster tree, we do not constrain the number of SAM iterations, i.e. using the same termination as SAM, which guarantees that TSAM and SAM converge to the *exactly same* minimum.

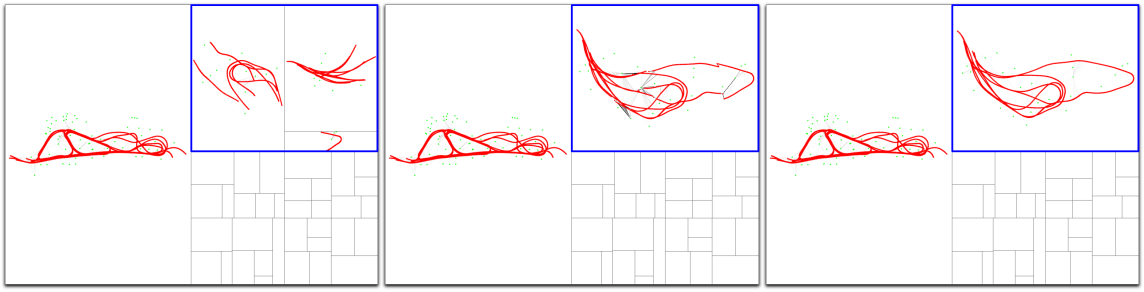


Figure 15: **The optimization of one submap in the Victoria Park data-set.** *The rectangle outlined in blue corresponds to the current submap. The intensity of the black lines indicates the amount of residuals on the corresponding measurements. From left to right: 1). Three child submaps that have been optimized before. 2). The new submap is created by aligning three child submaps using base nodes. Note that the three submaps are roughly aligned together, and a few constraints (black lines) are not satisfied very well. 3). The submap after the full optimization step. Nearly all the constraints are perfectly satisfied now.*

2.5 Summary

In Chapter II, we introduced the main process of the TSAM algorithm. As a divide-and-conquer approach, the algorithm clearly consists of the two steps, namely the divide step and the conquer step. The conquer step for linear systems is straightforward and is underlying graph elimination, while the conquer step for nonlinear system is rather more complicated. In the last section, the initial submap alignment is done together with the traditional optimization way, and we will introduce a much more efficient and robust way to improve the submap alignment process.

Chapter III

THE PROPERTIES OF TSAM

In this chapter, we will show the properties of TSAM and why TSAM has those advantages. First, we introduce the base nodes to move submaps rigidly, which speeds up the nonlinear optimization considerably. Second, we show that the smoothing steps employed in each subtree makes the corresponding submap fully optimized. Third, the submap initialization is incrementally generated following the cluster tree structure, and it has much better quality than that generated in the traditional ways. At last, we analyze how the tree structure in TSAM enables parallel and out-of-core computation, which makes TSAM a true scalable solution for the SLAM and SfM problems.

3.1 TSAM is Fast

3.1.1 Subtree Alignment with Base Nodes

TSAM solves the submap alignment problem by introducing base nodes and exploiting the local structure of the submaps. For each submap $\mathscr{M}_k$, we assign a base node b_k that represents the position and the orientation of the submap in its parent's coordinate system. Considering the structure of the cluster tree, if the submaps between the submap $\mathscr{M}_k$ and the root are $\mathscr{M}_p, \mathscr{M}_{p+1}, \cdots, \mathscr{M}_{k-1}$, we may see that a robot pose x_i in the submap $\mathscr{M}_k$ satisfies the following relationship:

$$x_i = b_p \otimes b_{p+1} \otimes \cdots \otimes b_{k-1} \otimes b_k \otimes \widetilde{x}_i$$

where $\widetilde{x}_i$ is the robot pose x_i in the local coordinate system of submap $\mathscr{M}_k$. The similar chained rigid transformation also applies to all the landmarks in submap $\mathscr{M}_k$.

In this way, the resulting optimization process only works on $F_{\mathscr{M}}$ and the base nodes of the child clusters, as shown in Figure 16. The variable set $C_{\mathscr{M}}$ will stay fixed as anchor

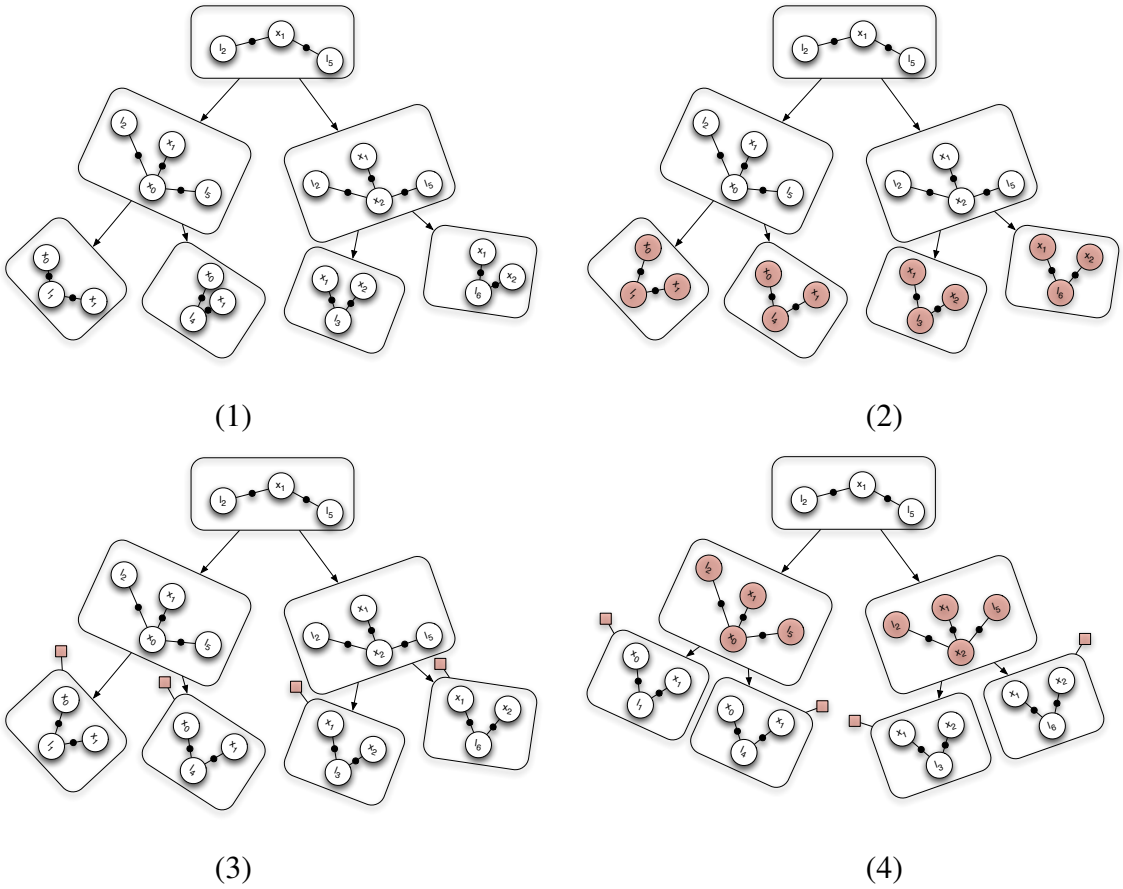


Figure 16: **The nonlinear optimization for the cluster tree created in Figure 12.** *(1). The noisy initialization of the cluster tree; (2). The full optimization in the leaf submaps; (3). Four base nodes are assigned to the leaf submaps to represent their rigid transformations with respect to the parent submaps; (4). The second-level submaps are optimized together with the four base nodes. The remaining figures continue in Figure 17.*

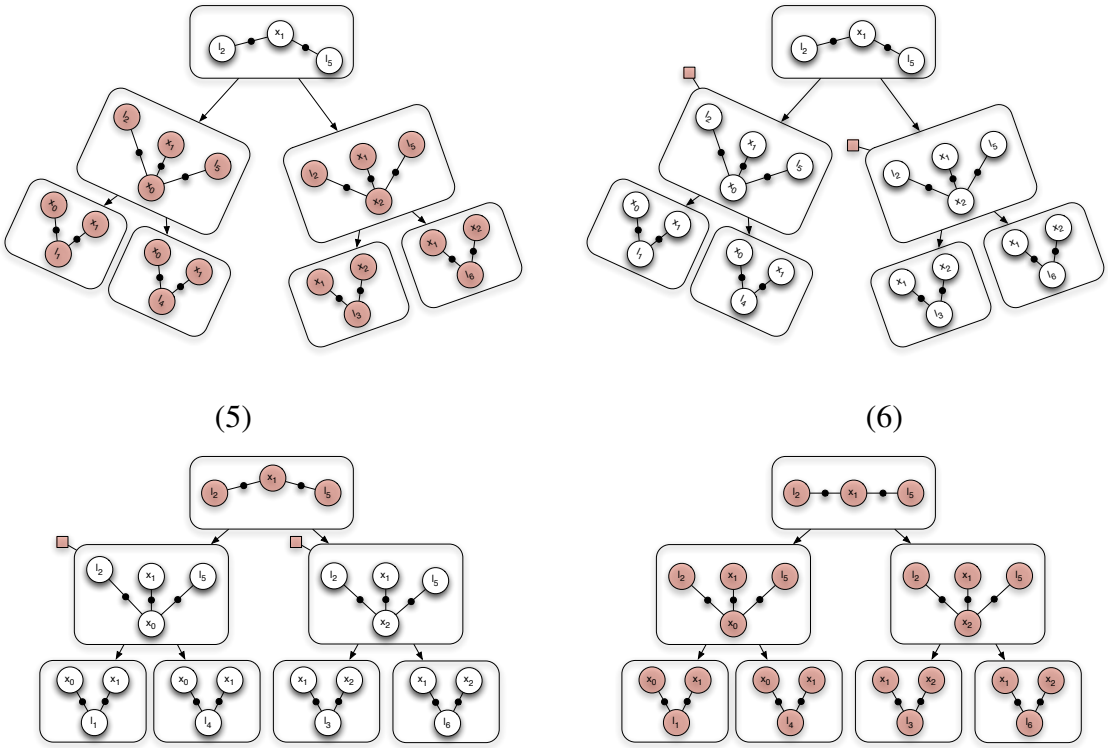


Figure 17: **Continue on the nonlinear optimization for the cluster tree created in Figure 12** *(5) Full optimizations in two subtrees; (6) Two base nodes are assigned to the subtrees; (7) The root submap is optimized together with the two base nodes; (8) A full optimization on the entire graph.*

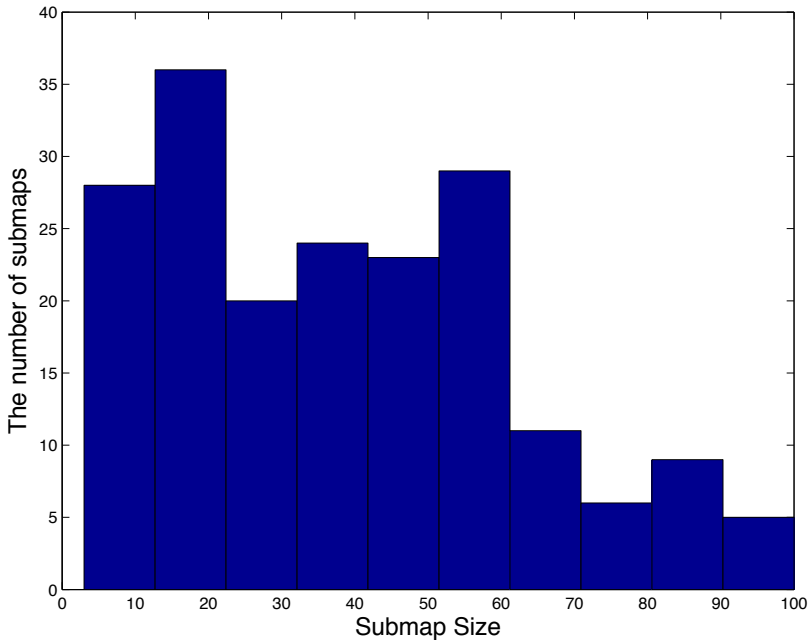


Figure 18: **The histograms of submap sizes for the Victoria Park data-set.**

variables (labelled by anchor icons in the figure). Note that only the base nodes of direct child submaps are movable (b_2 and b_3 in Figure 16), since the base nodes of other child submaps (b_4, b_5, and b_7) have already been optimized over when optimizing maps $\mathscr{M}_2$ and $\mathscr{M}_3$.

The complexity of the new submap optimization is greatly reduced with respect to the naive approach. The complexity of the method without base nodes is $O(|C_{\mathscr{M}} \cup F_{\mathscr{M}}|^2)$. By introducing the base nodes, we limit the complexity to $O((|F_{\mathscr{M}}|+s)^2)$ considering that there are s base nodes involved. In the experiments, we will show that $|F_{\mathscr{M}}|$ is usually much smaller than $|C_{\mathscr{M}}|$ and almost stays constant due to the minimized sizes of vertex separators. On the other hand, $|C_{\mathscr{M}}|$ increases with respect to the number of clusters in $\mathscr{T}_{\mathscr{M}}$. For the large-scale SLAM problems, $|C_{\mathscr{M}}|$ of the submaps close to the root is almost equal to the number of total variables and consequently dominates $|C_{\mathscr{M}} \cup F_{\mathscr{M}}|$.

3.1.2 Experimental Results for Timing Tests

We first tested our algorithm on the public Victoria park data-set, which contains about 6900 laser scans with corresponding odometry readings. All the results reported in this

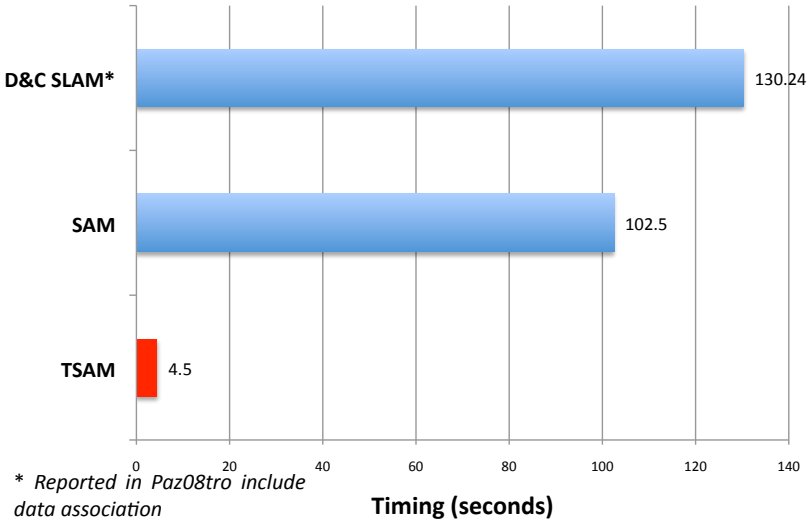


Figure 19: **The comparison of timing results on Victoria Park data-set between TSAM, SAM and D&C SLAM.**

thesis were produced on a Macbook Pro with 2.8GHz CPU.

As shown in Figure 5, the SLAM graph is recursively partitioned into multiple submaps, and one of the submap optimization steps is illustrated in Figure 15. The advantage of using a nested dissection based recursive partitioning is shown in Figure 18. We can see that, despite having more than seven thousand nodes in the entire graph, the sizes of most submaps are between 10 to 60. In fact, the average size of all the submaps is 38.3, which is only 0.52% of the total size. The root submap, i.e. the vertex separator between three coarsest-level submaps only contains 19 nodes. Given those 19 nodes, the three submaps are statistically independent. When assembling those submaps, we only need to optimize over those 19 nodes as well as the three base nodes.

In terms of efficiency, we compared our algorithm with SAM [19] and D&C SLAM [81], which is one of the fastest submap based algorithms available. As the initial estimate is quite off due to the noisy measurements, we found that SAM does not converge if directly applied to the entire map. Hence we gradually increased the size of the active map and applied the algorithm sequentially. Also note that the timing of D&C SLAM was originally reported in [81] with 2.8GHz CPU and did include data-association overhead. The final

Figure 20: **TSAM applied to the Intel lab data set.** 739 *robot poses represented by red dots are overlaid on the map.*

comparisons are plotted in Figure 19. TSAM was able to finish the entire optimization in less than 5 seconds.

We also evaluate our algorithm in a popular pose SLAM data set, namely the Intel lab data-set, as shown in Figure 20. The residuals from TSAM and SAM are both 0.051. The timing results of TSAM in Figure 21 is compared with the results reported in [78] as well as those from TORO [44] and HOG-Man [40]. Note that SAM converges very fast on this data-set due to the relatively low noise level in the data set.

3.2 TSAM is Exact

The ability to produce accurate maps and pose estimates is one of the most crucial factors for autonomous robot applications, as the inaccuracy of the map or the robot pose estimate could greatly hurt the performance in many scenarios. So far we have introduced the main process of TSAM algorithm, but the focus has been on improving the efficiency. One question to ask at this point is whether we make any approximation in the algorithm. In this section, we will explain why TSAM is an exact algorithm for the SLAM problem and

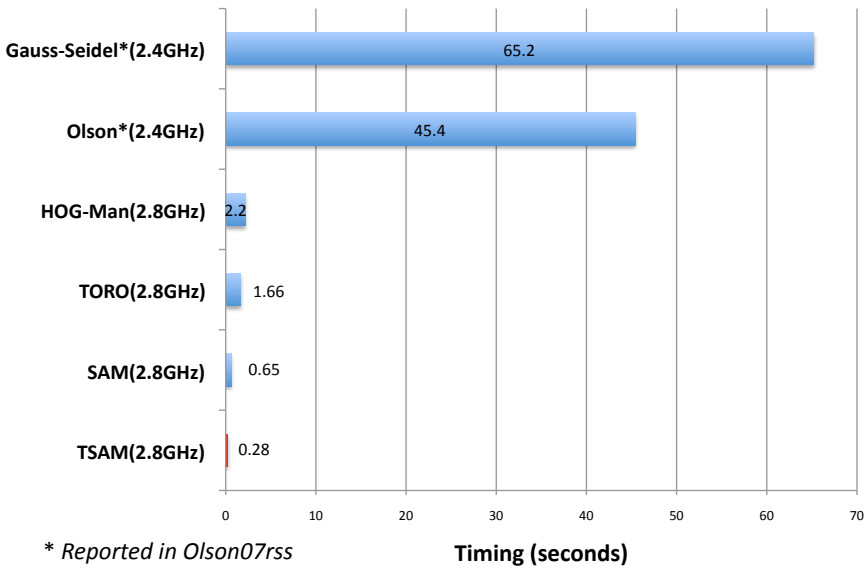


Figure 21: **The comparison of timing results on the Intel lab data-set.**

the SfM problem.

3.2.1 Related Work

The SLAM algorithms can be mainly grouped into two classes. The first class is the filtering base approaches, including extended Kalman filters [61, 88], sparse extended information filters [25, 93], and particle filtering SLAM [71, 72], etc. However, due to the linearization error, the filtering based approach is like to become inexact [49]. Smoothing is the other class of approaches [69, 70, 45, 34, 19, 50], in which the entire robot trajectory is considered rather than the latest pose. The goal of these approaches is computing the maximum likelihood estimate. Among them, some approaches [69, 70, 45] do smoothing only on the robot trajectory, while the others [34, 19, 50] consider the full SLAM problem.

Sensor measurements such as GPS may also supply some global constraints to the SLAM problem hence greatly improve the accuracy of the estimate. Kummerle et. al. [60] employed aerial images as prior information and greatly improve the quality of the map in the large-scale SLAM problem.

One canonical way to do exact nonlinear optimization on the SLAM problem is Smoothing and Mapping [19], which addresses the full SLAM problem by optimizing all the variables (robot poses and the landmarks) at once. As long as the initialization is enough close to the global minimum point, SAM guarantees the optimization to converge the global minimum as well as finding a maximum likelihood estimate.

3.2.2 Exactness in TSAM

After partitioning using the nested dissection algorithm, Tectonic SAM employs the cluster tree [57] as the main data structure for the further inference. The cluster tree is a well-known graphical model for exact and fast inference, which has two properties. The first property is the *family preservation property*, which indicates that all the factors (measurements) in the original graph are encapsulated by the corresponding cluster tree. This is one of the key reasons why TSAM is able to solve the SLAM problem and the SfM problem exactly, as all the nonlinear measurements are exploited during the optimization. The second property of the cluster tree is the *running intersection property*, which indicates that if a variable exists in two clusters, then the same variable also exists in all the clusters on the path that connects those two clusters.

Tectonic Smoothing and Mapping is directly derived from Smoothing and Mapping, whose exactness has been proved in [19]. The major difference between the TSAM nonlinear optimization and the SAM approach is that we use the base nodes to speed up the convergence, but the exactness does not change. For each subtree, the optimization is done in two steps. In the first step, the variables in the root cluster of the subtree are optimized together with the base nodes of the first-level submaps, as illustrated in Figure 23. Note that all the variables in the child clusters stay fixed in this step. In the second step, a full optimization is employed to balance all the variables. If we skip the first step, the nonlinear optimization still works but will take more time than the base-node version. Note that this step is equivalent to invoke Smoothing and Mapping on the subtree variables, as shown in

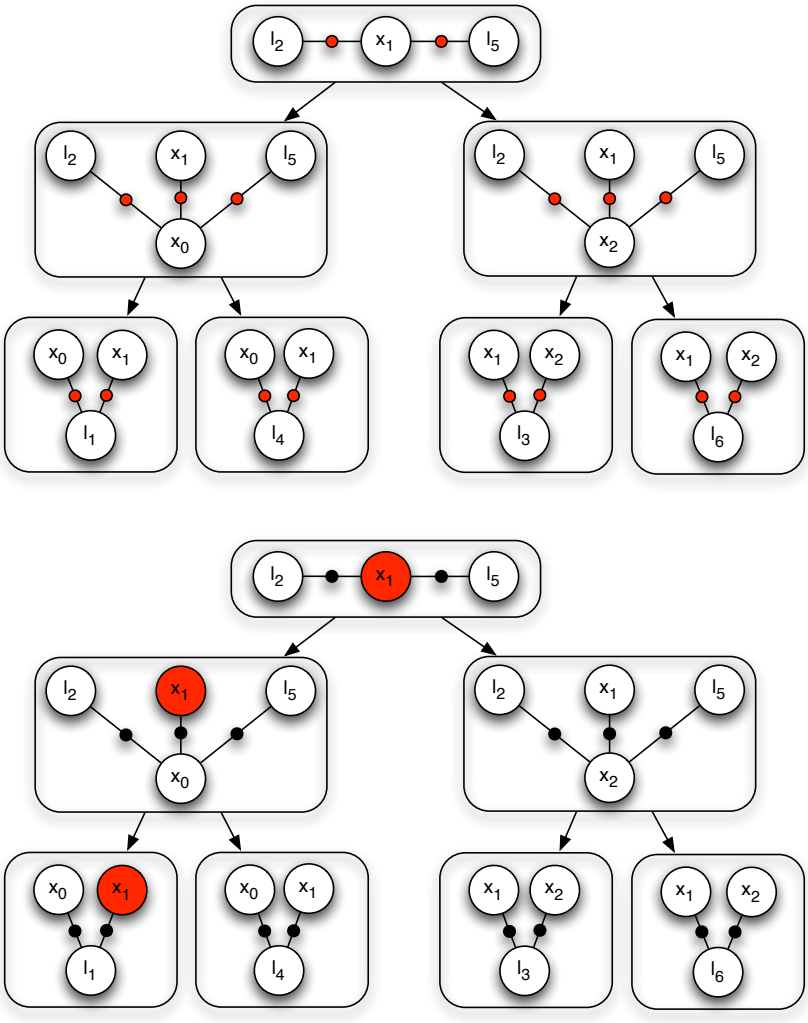


Figure 22: **The two properties of the cluster tree.** *Top: The family preservation property. All the red dots represent the factors in the original factor graph; Bottom: The running intersection property. Node x_1 exists in all the clusters on the highlighted path.*

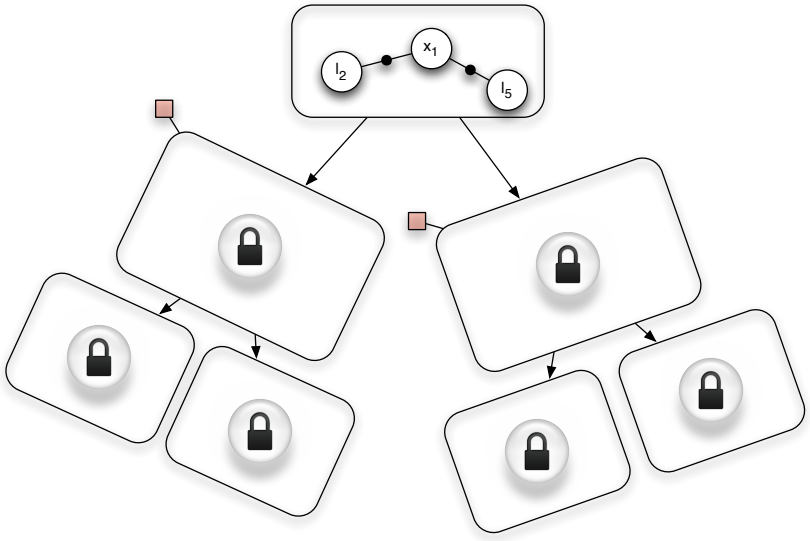


Figure 23: **The first step of TSAM nonlinear optimization aligns the submaps using their base nodes.** *All the variables inside the child submaps are locked. The base nodes (red squares) are optimized together with the variables in the parent cluster. All the variables in the child clusters are locked as they have been previously optimized in the bottom-up process.*

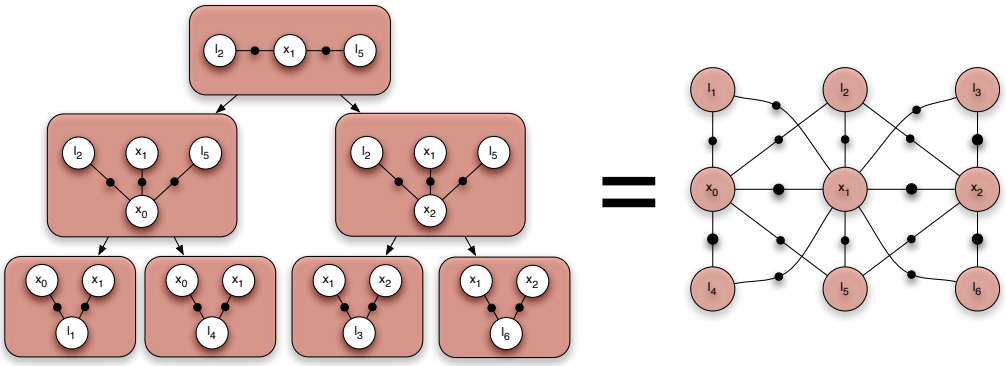


Figure 24: **The second step of TSAM nonlinear optimization balances all the variables in the current subtree.** *As the full optimization step involves all the variables and the factors(measurements) in the factor graph, it is indeed equivalent to Smoothing and Mapping [19].*

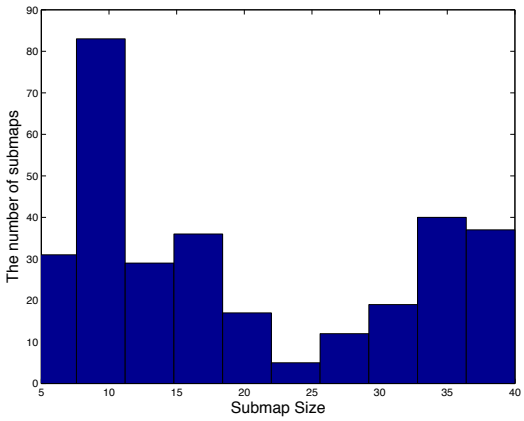


Figure 25: **The histograms of submap sizes in the block-world simulation.**

Figure 24.

3.2.3 Experimental Results for Exactness Tests

In the previous experiments on the Victoria Park data set, we observed that TSAM converged to *exactly* the same minimum as SAM. In the full optimization steps of non-root clusters, our algorithm ran two iterations of full subtree optimization. For the root submap, it took one additional iteration, i.e. 3 iterations of the full optimization, to converge to the optimal point. This result verified that TSAM is *exact*.

To further verify the performance of the proposed algorithm, we created a block-world simulation that contains a trajectory with 2640 robot poses and 3200 landmarks, as illustrated in Figure 26. We observed similar submap sizes as those found on the Victoria Park data-set, as shown in Figure 25. Since the map of the block-world simulation is more structured than the map of Victoria Park, the submap sizes are more consistent. Overall in both data-sets, the sizes remain small compared to the total map size. For example, the root submap contains 29 nodes, i.e. 0.50% of the total nodes. Having such small vertex separators is the key for our algorithm to achieve high efficiency. We refer interested readers to [58] for more theoretical proofs.

As we know the ground truth of the simulation data, we again verified that both TSAM and SAM converged to the *same* minimum point. We also compared the efficiency of our

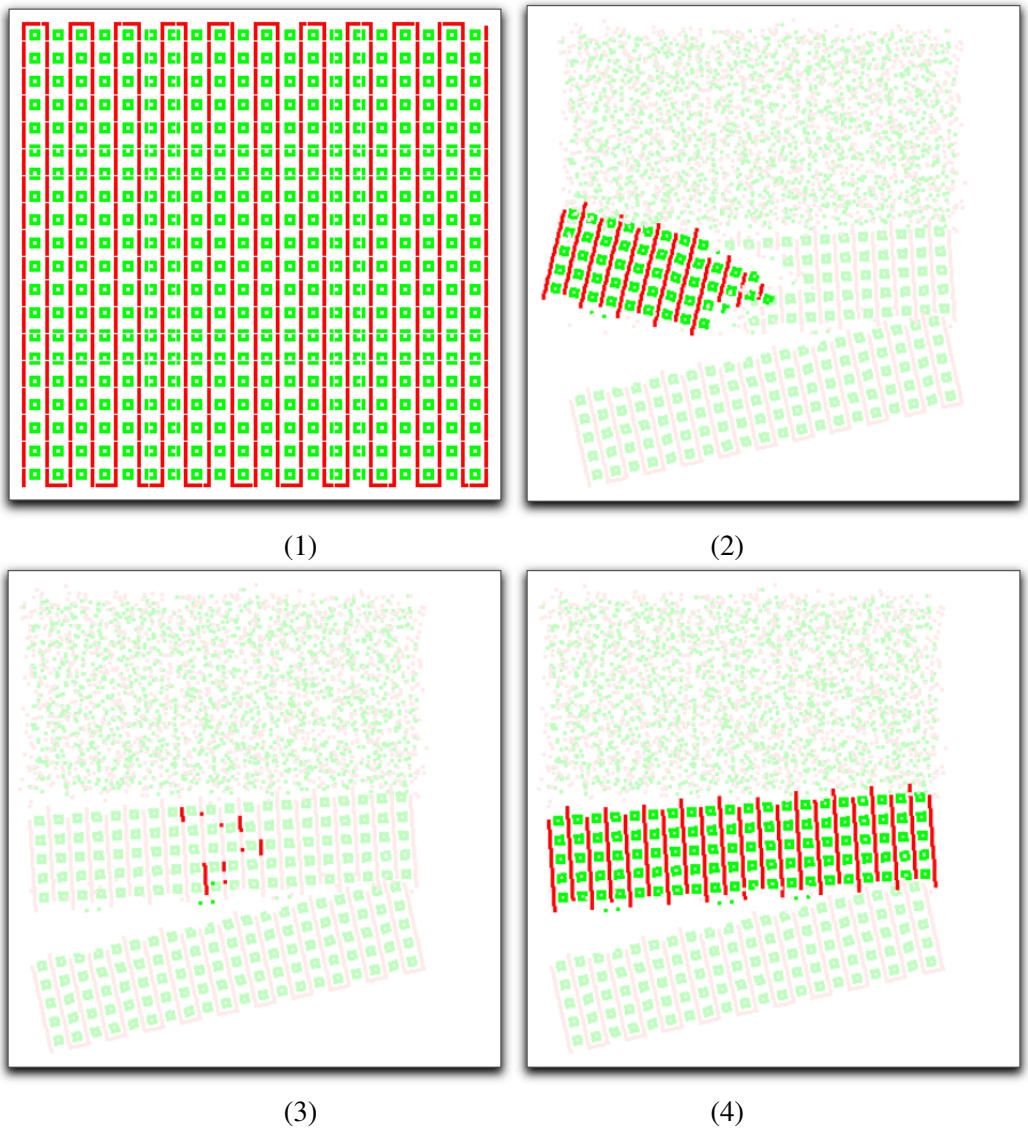


Figure 26: **The block-world simulation in which the robot poses are rendered in red and the landmarks are rendered in green.** *The high-lighted nodes are the variables being optimized, and the other inactive nodes are rendered in the washed-out colors. (1). The original block-world simulation. (2). The optimization of a certain submap $\mathscr{M}_k$. (3). The high-lighted part is the vertex separator between submap $\mathscr{M}_k$ and another submap $\mathscr{M}_{k+1}$ on the right. (4). The one-iteration full optimization for the quarter-size map. Note that there is no visible change between the third figure and the fourth figure, hence one iteration is sufficient to re-balance all the nodes in the quarter-size map.*

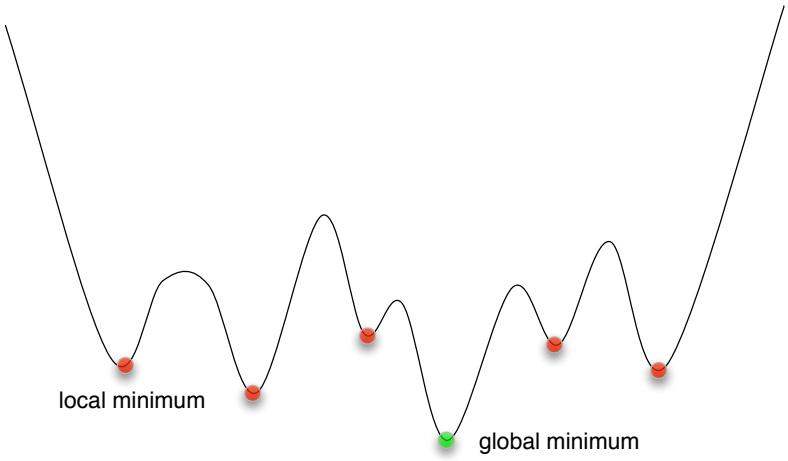


Figure 27: **The nonlinear nature of the SLAM problem.** *There are multiple local minima but only a single global minimum. The initialization point becomes crucial for the correct convergence of the nonlinear optimization method.*

algorithm to SAM. For each submap including the root submap, TSAM ran two iterations of full optimization. TSAM took 13.6 seconds to converge, while SAM took 67.9 seconds to finish.

3.3 TSAM is Robust

3.3.1 Incremental Initialization

When talking about the *robustness* of an algorithm for the SLAM or SfM problems, there are typically two issues to consider. First, how well the algorithm behaves with regard to the noise level in the measurements. In other words, how the performance changes when the measurements are contaminated by a large amount of noise. Second, given noisy initialization, whether the algorithm is able to recover the global minimum after the nonlinear optimization. In this chapter, we will show that TSAM is able to tackle both of the problems.

Due to the nonlinear nature of the SLAM problem and the SfM problem, as illustrated in Figure 27, special consideration must be taken to avoid the local minima during the nonlinear optimization. The key to solve this problem is to initialize the optimization properly. Incremental SLAM approaches [71, 72, 50] typically have the advantage of obtaining better

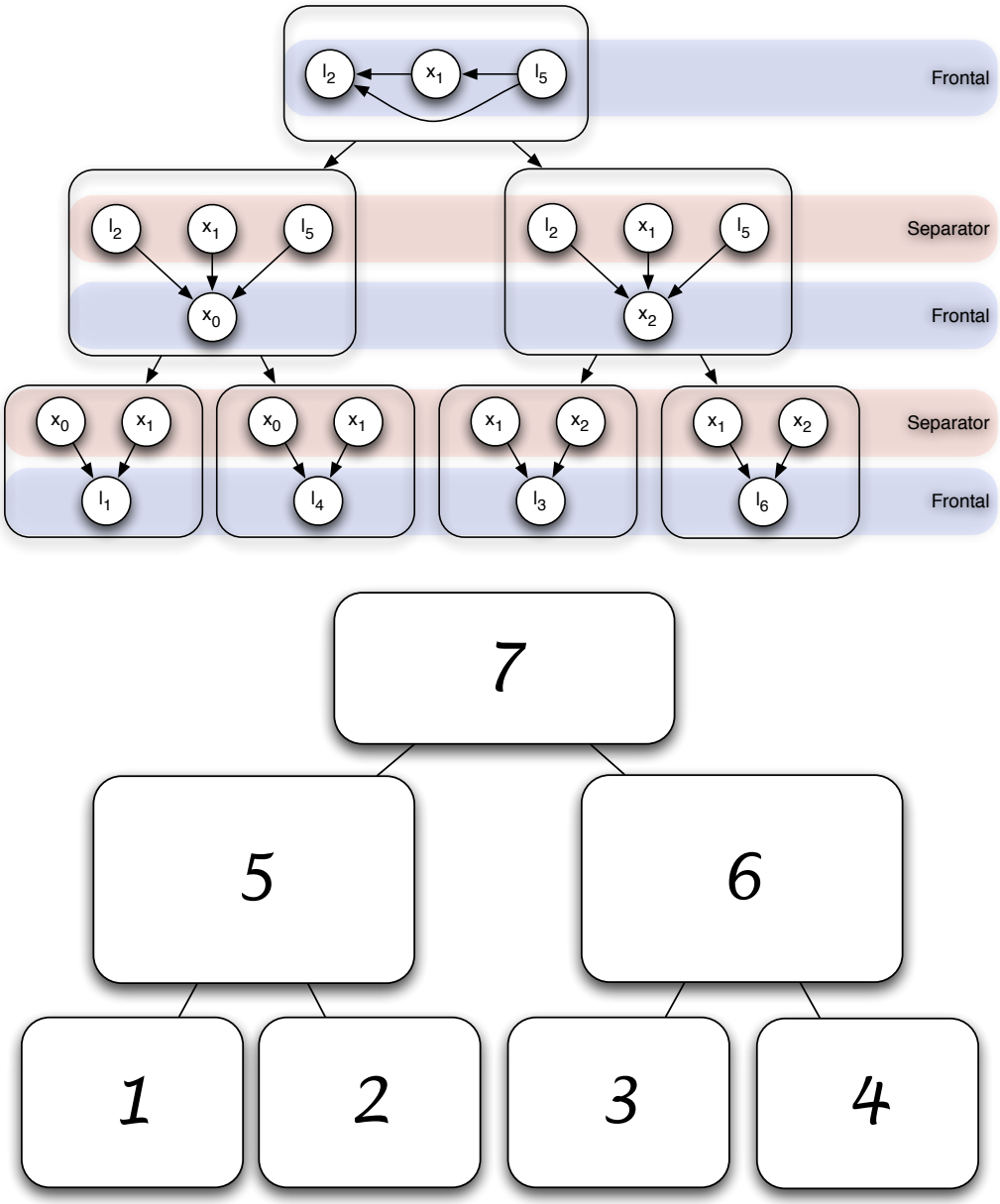


Figure 28: **The incremental initialization of the TSAM algorithm.** *Top: The cluster tree to be optimized. Bottom: All the clusters in the cluster tree are optimized in the order of one to seven, such that the initialization of the parent cluster can be built using the latest estimate of the child clusters. For example, by computing the pose of the base nodes of cluster* 1 *and cluster* 2*, all the variables in those two clusters can be transformed to the coordinate of cluster* 5*. Combined with the separately initialized variables in cluster* 5*, the entire initialization of the left subtree is obtained.*

linearization, because the estimate is built gradually until the entire problem is solved.

The divide-and-conquer scheme also enable us to obtain a good initialization to batch approaches, which is one of the most crucial issues in nonlinear optimization. One way to achieve good initialization is to use incremental approaches. Snavely [90] employed an approach similar to that in [10] to build a photo tourism system enabling users to travel in a large virtual 3D world. Klopschitz [56] improves the robustness of the incremental SfM algorithm by integrating more stable image triplets. Compared to incremental approaches, typical batch approaches suffer from the bad initializations, e.g. those computed by composing robot odometries. By employing a divide-and-conquer approach, we can recursively compute the initializations from the optimized submaps, which are much accurate than those from conventional approaches.

Although Tectonic SAM is a batch approach, it employs the same incremental initialization scheme as the incremental approaches, as shown in Figure 28. The cluster tree generated by the nested dissection algorithm, i.e. the hierarchical structure of the partitioned submaps, lends the ability of building the initialization from scratch. More specifically, the initialization is computed by integrating the latest estimate from child clusters in a bottom-up fashion. As illustrated in Figure 16, the TSAM optimization for each cluster has two steps. In the first step, the base-node poses of its child clusters are computed during the optimization. Hence the rigid transformations of its clusters are recovered and can be used to transform all the variables in the child clusters to the same coordinates as the parent cluster, and the resulted map serves as the initialization point of the full optimization of the subtree.

3.3.2 Experimental Results for Robustness Tests

In this section, we introduce four sets of simulation data to evaluate the performance of both the TSAM algorithm and the SAM algorithm when dealing with noisy measurements. The four simulations are as follows: straight data-set(Figure 29.(1)), loop data-set (Figure

Table 1: **The sizes of four simulations: straight, loop, spiral, and block-world.**

Simulation	Pose Nr.	Landmark Nr.	Measurement Nr.
Straight	5000	10000	24998
Loop	500	4974	20500
Spiral	211	538	8650
Block-World	378	392	10291

29.(2)), spiral data-set (Figure 30.(1)), and block-world data-set (Figure 30.(2)).

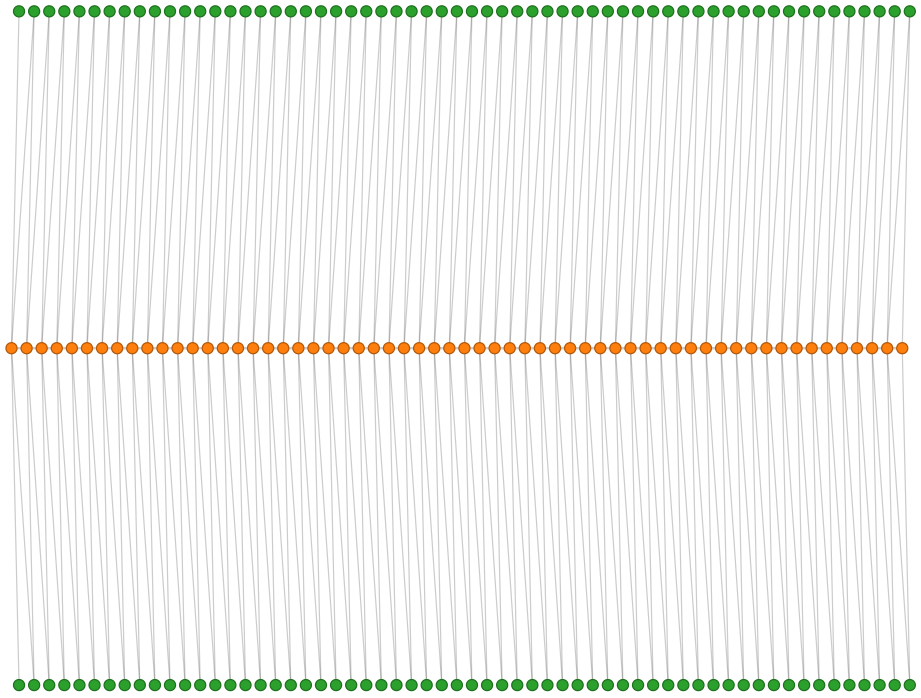
3.3.2.1 Simulations

In the simulation experiments, we fix the number of variables in the SLAM problem but vary the noise levels of the measurements, i.e. adding Gaussian noise with different standard deviations to the raw measurements computed using ground-truth data. For all the four simulations, we summarize the results by averaging the results from 20 runs, and use the SAM [19] implementation for comparison in all the experiments. The scales of all the simulations are listed in Table 1.

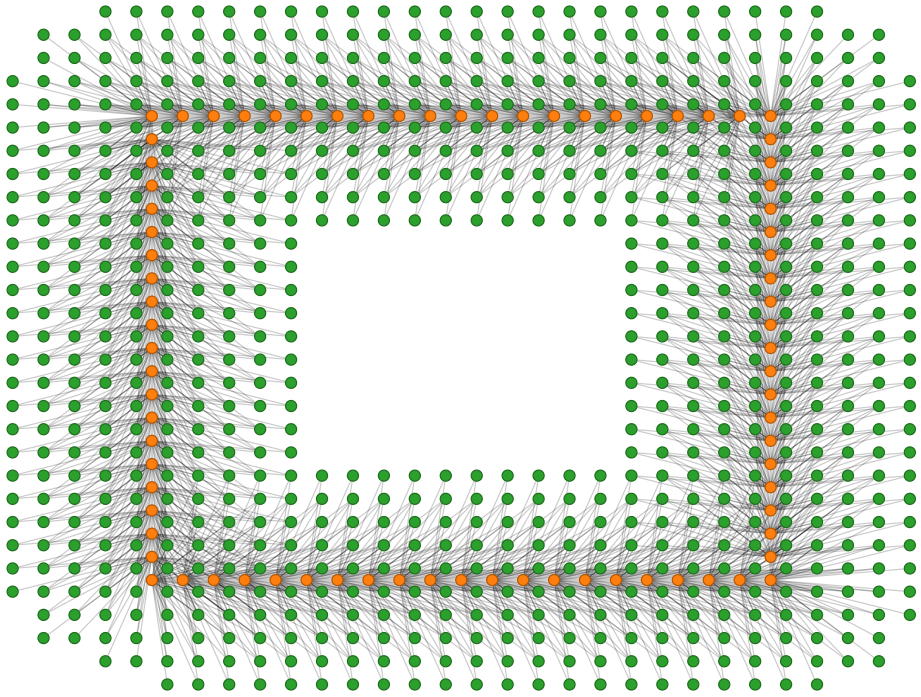
The timing cost consistently increases when the noise level goes up, but the trends are different for the four simulations. In the straight simulation and the loop simulation, SAM is only slightly expensive than TSAM. On the other hand, in the more loopy experiments, the increasing noise greatly confuses the nonlinear optimization, and SAM becomes much more expensive in those scenarios, as shown in Figure 31.

Due to the fact that the number of variables stays fixed, the time per iteration remains relatively stable in all the four simulations (Figure 32). The single dominant factor for SAM's timing cost is the number of iterations, i.e. more noise means more iterations until convergence, as shown in Figure 33. Note that there is always a small bump when the noise goes up to a certain threshold, which is consistent with the quadratic assumption of the nonlinearity. More specifically, the convergence is a lot faster when the state variables stays in the quadratic bowel of the nonlinear manifold.

The break-down timing of TSAM (Figure 34) shows that the partitioning is a small

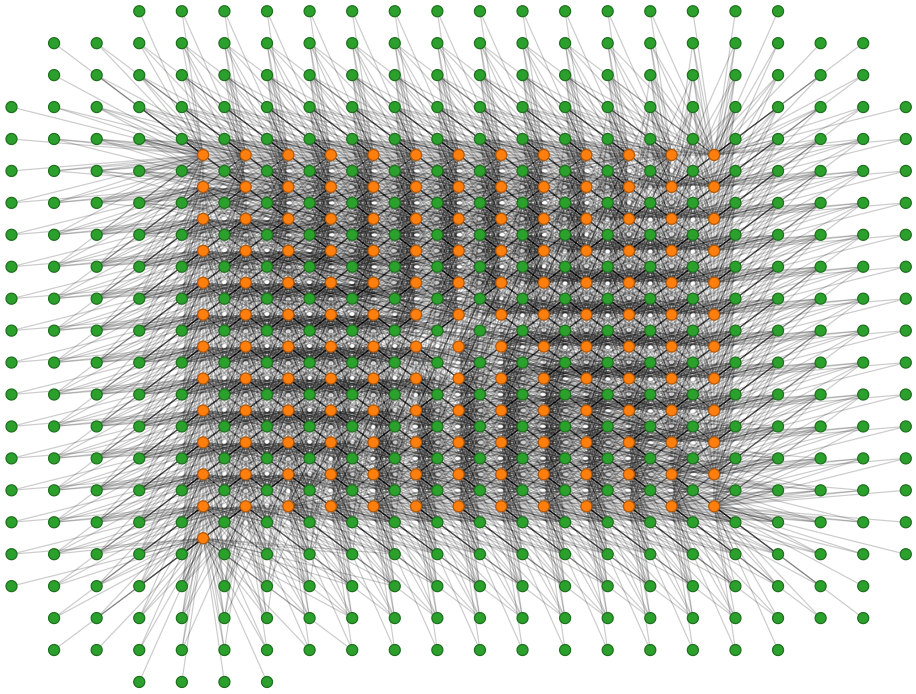


(1)

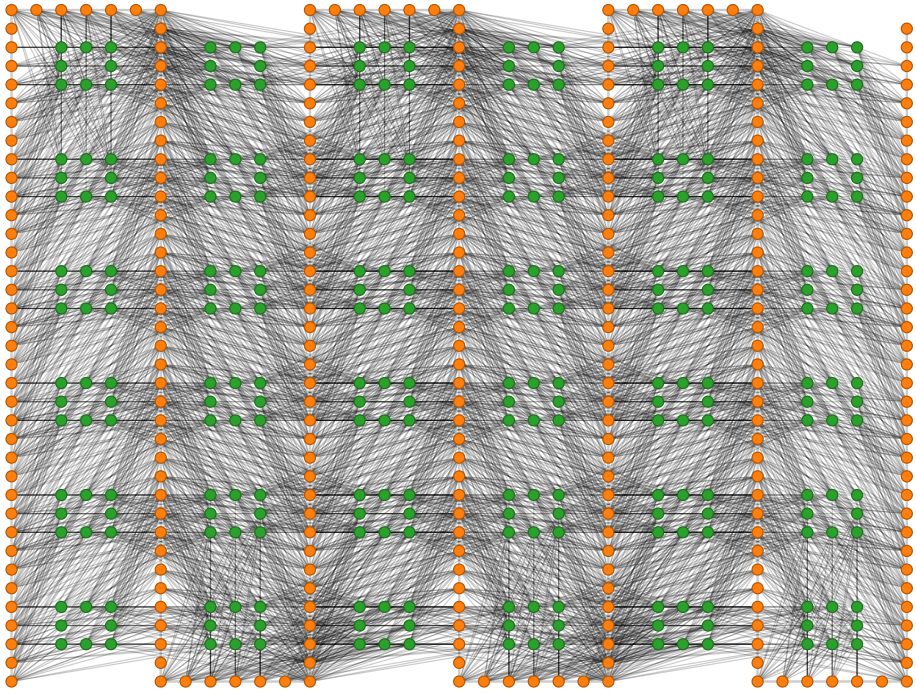


(2)

Figure 29: **The simulation of moving straight and moving for one loop.** *The orange dots represent the robot trajectories, and the green dots represent the landmarks. The gray lines represent the landmark measurements.*



(1)



(2)

Figure 30: **The simulation of moving in a spiral shape and moving in a block-world.** *The orange dots represent the robot trajectories, and the green dots represent the landmarks. The gray lines represent the landmark measurements.*

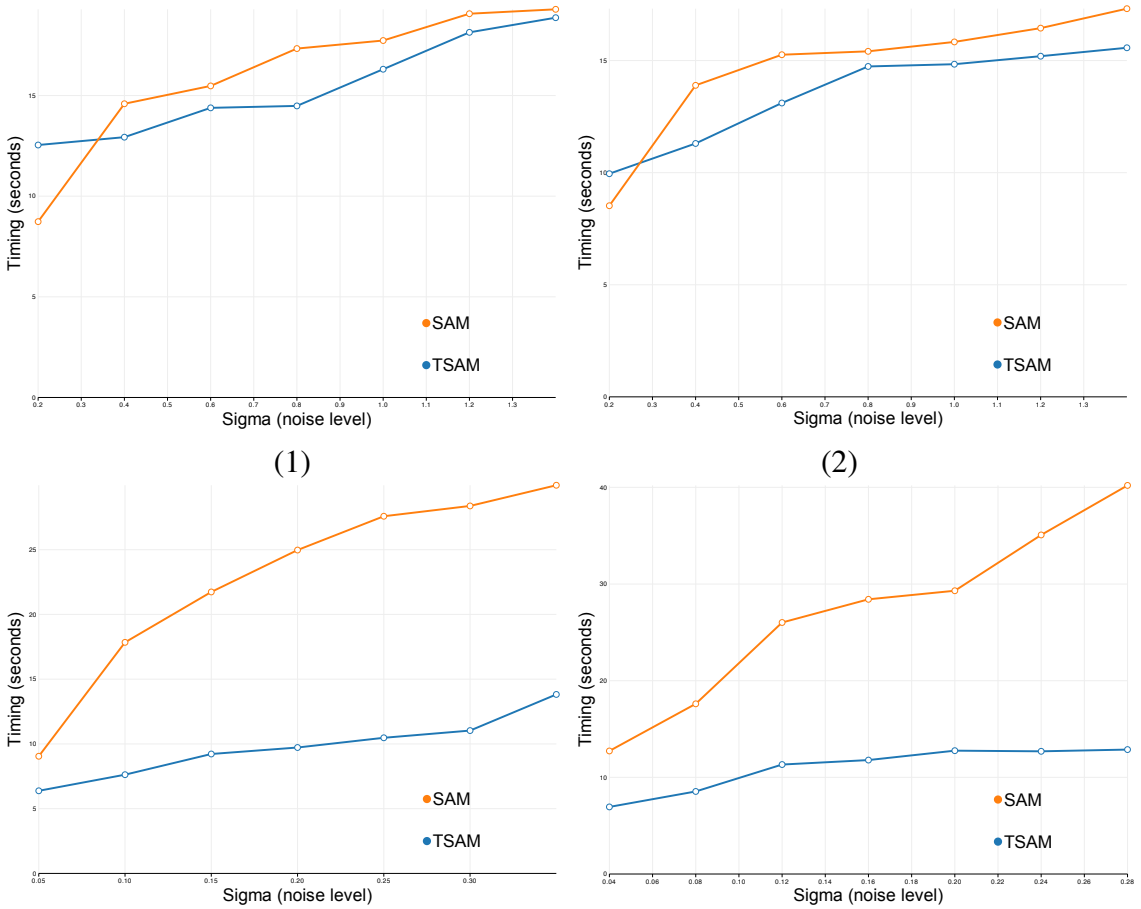


Figure 31: **The noise level versus the timing when applying SAM and TSAM to the four simulations**: *(1) straight; (2) loop; (3) spiral, and (4) block-world data sets.*

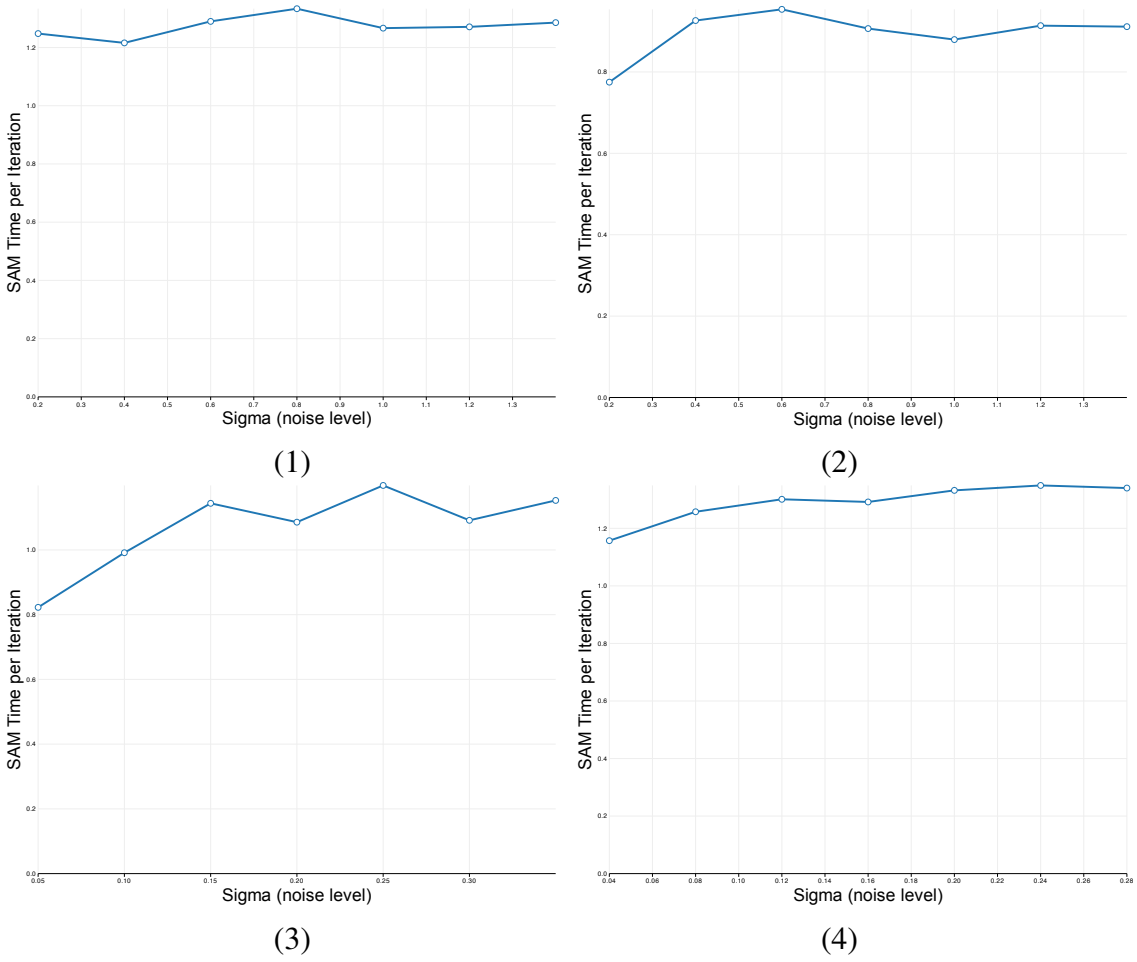


Figure 32: **The noise level versus the timing per iteration when applying TSAM to the four simulations**: *(1) straight; (2) loop; (3) spiral, and (4) block-world data sets.*

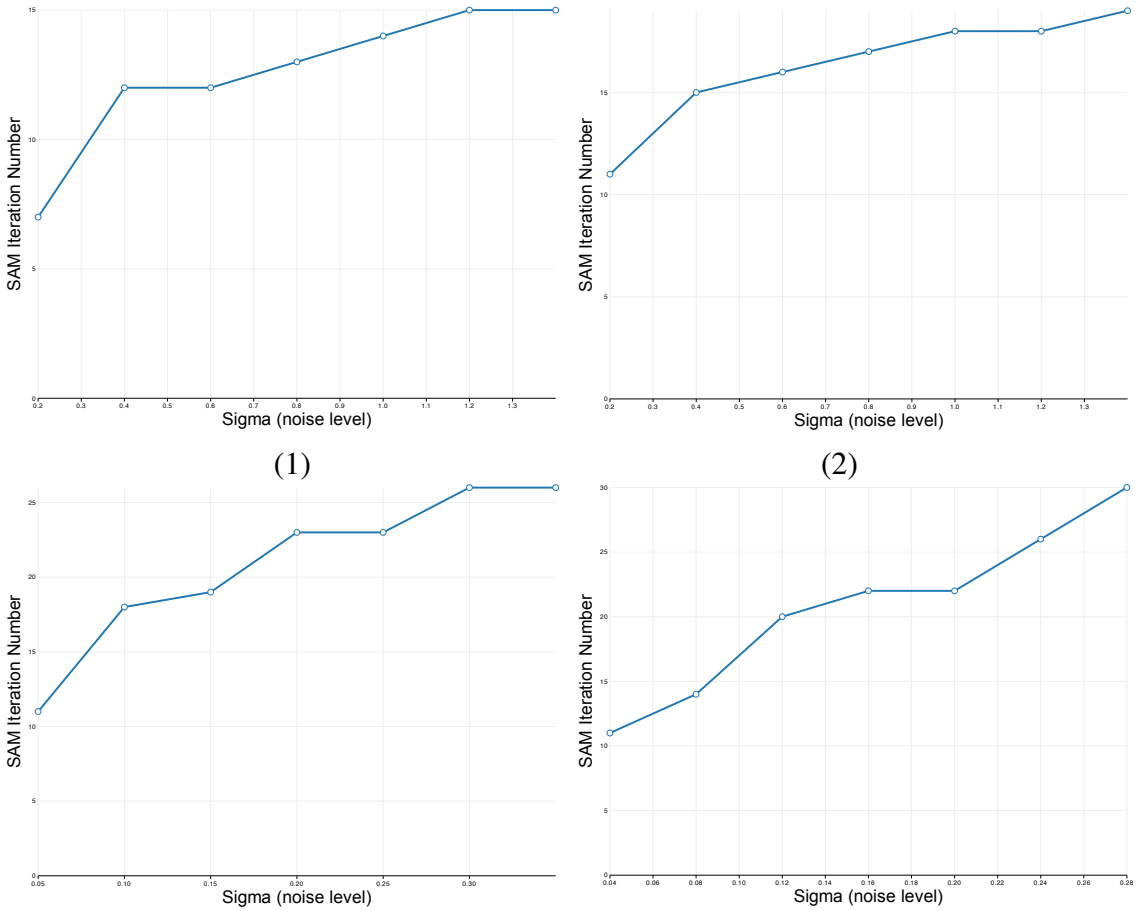


Figure 33: **The noise level versus the total number of iterations when applying SAM to the four simulations**: *(1) straight; (2) loop; (3) spiral, and (4) block-world data sets.*

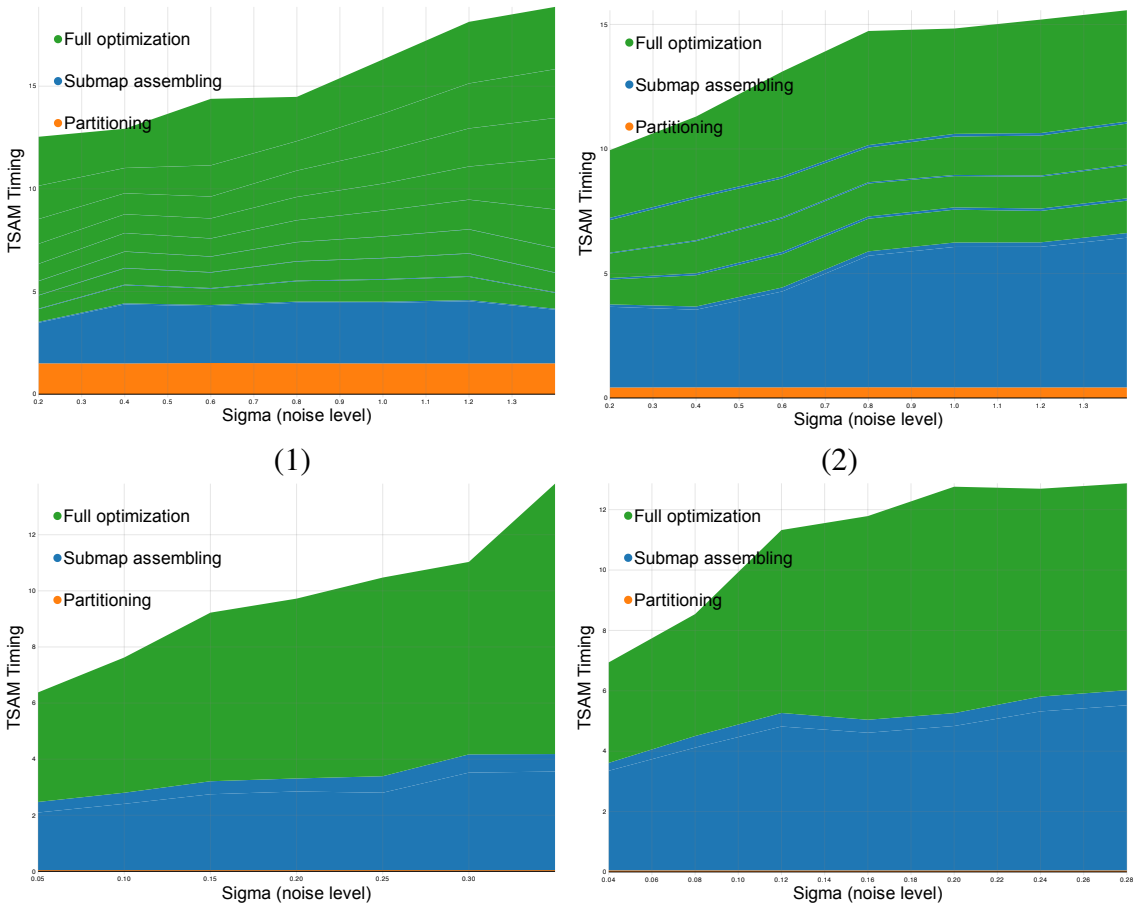


Figure 34: **The noise level versus the timing break-down in each TSAM stage when applying TSAM to the four simulations**: *(1) straight; (2) loop; (3) spiral, and (4) block-world data sets.*

overhead especially when solving challenging problems. The submap assembling uses relatively constant time due to the fact that this step is only for make the system stay relatively close to the global minimum. On the other hand, when the noise goes up, the full optimization time goes up, which means it does become more expensive to rebalance all the variables. However, such change is still much smaller compared to that in SAM's timing results (the green areas in the last column looks large because they are scaled according the TSAM's timing results).

3.3.2.2 Victoria Park Dataset

To verify the robustness of the TSAM algorithm when dealing with noisy initialization, we do the second experiment based on Victoria Park data-set. The initialization is generated using the raw measurements that come with the original data-set. As we can see from Figure 35, the global optimization is sensitive to the initialization point, and SAM converges to an incorrect local minimum after starting from the bad initialization point.

In contrast, Tectonic SAM does not need any pre-determined initialization point. Instead, the algorithm computes the map from scratch and gradually extends the nonlinear optimization to the entire map. Such an incremental initialization not only helps the TSAM algorithm converge faster but also recover the correct map, as shown in Figure 35.(2).

3.4 TSAM is Scalable

How to make the nonlinear optimization scalable is one of the most challenging problems when moving towards large-scale outdoor SLAM/SfM [90, 35, 2, 31]. Different numerical methods for the nonlinear optimization, especially the matrix factorization methods, have been explored by the researchers. Dellaert and Kaess introduced an efficient algorithm to solve the linearized problems by exploiting the sparsity of the Jacobian matrix with a good variable elimination ordering[19, 50]. PhotoSynth [90] tackles the bundle adjustment problem using the Sparse Bundle Adjustment (SBA) package [67], which also exploits the sparsity pattern underlying the SfM problem.

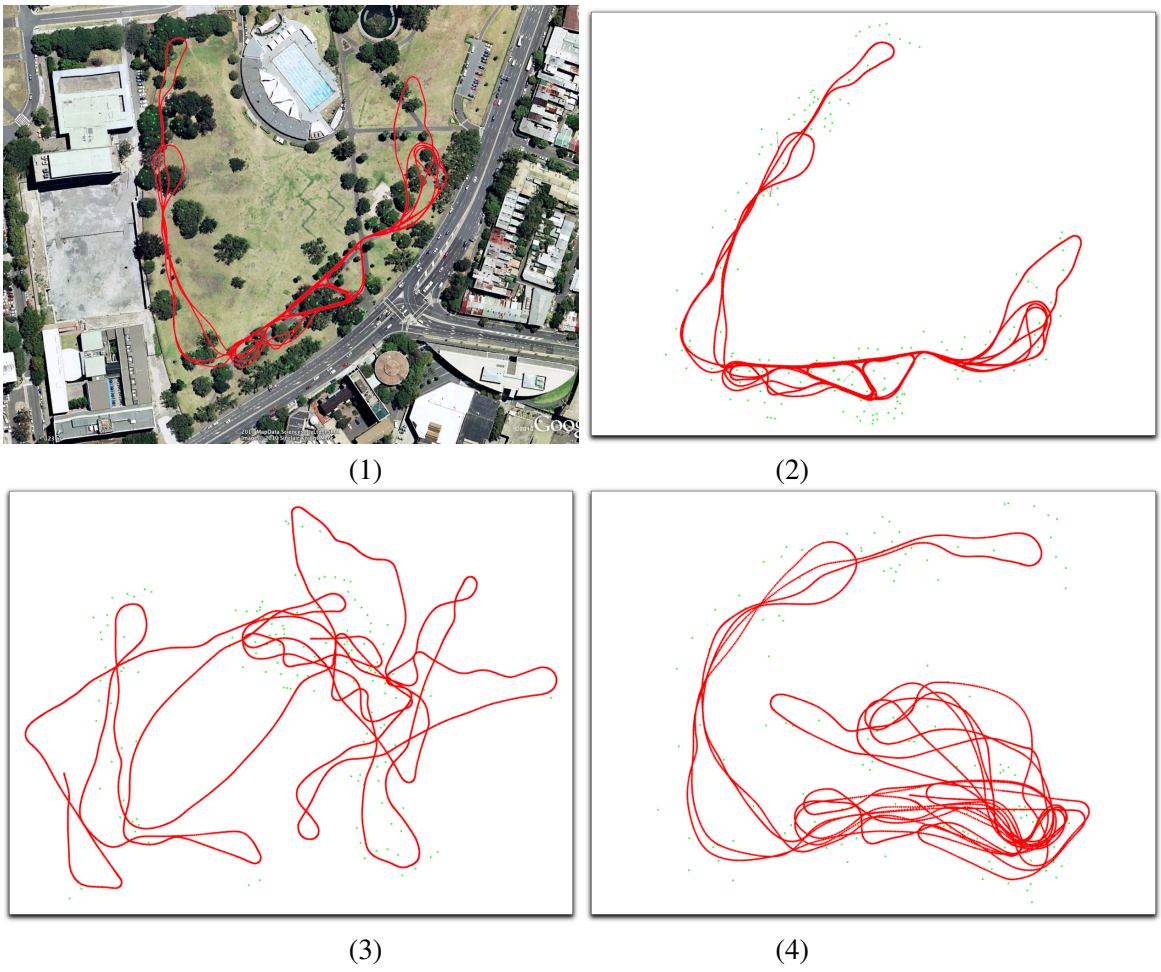


Figure 35: **How the initialization point influences the final map.** *(1). The ground truth landmarks (green) and the robot trajectory (red) overlaid on the aerial image; (2) The optimization results from TSAM does not need any pre-computed initialization; (3). The initialization computed from noisy measurements is of bad quality; (4) The incorrect map computed by SAM starting from the initialization point as shown in (3) .*

For large-scale urban environments, the sparse matrix factorization in traditional bundle adjustment are often still too big to fit into core memory. Therefore, more sophisticated techniques must be used. One option is to take a hierarchical, divide-and-conquer approach. For example, in both [27] and [76] the scene is partitioned into several smaller scenes that are easier to solve. Similar to [40], a skeletal graph idea [89] was also reported to solve SFM problems. Skeletal graph is helpful to obtain a good initialization since the optimized subgraph supplies a roughly optimal solution. The recursive partitioning approach of [9] is an early example of using a nested dissection in an aerial photogrammetry setting. In nested dissection, the parameter network is partitioned into several submaps. The submap parameters are grouped together and ordered first in the Hessian. Parameters associated with measurements that span submaps are called separator variables, and are ordered last. By ordering the variables in this manner, a standard sparse Cholesky factorization will compute the factorization of each submap first, followed by the factorization of the separator. Because submap variables do not have connections to variables in other submaps, the Cholesky factorization can be modified to compute the submap factorizations in parallel.

3.4.1 Parallel & Out-of-Core

When choosing a good algorithm to deal with large-scale SLAM/SfM data sets, two of the most appealing properties which people pursue are that the problems are *parallel* and *out-of-core*. Parallel is mainly about the constraints on the computation power and refers to the property that the algorithm can be executed one piece at a time on different computing resources, such as CPUs and servers, and the results can be merged later to obtain the same result as running a non-parallel algorithm. Out-of-core is mainly about the memory constraints and refers to the property that the original large problem that can not fit into a computer's memory at once can be solved by loading a small portion of the problem each time.

One advantage of submap based approaches is that the computation can be done in an

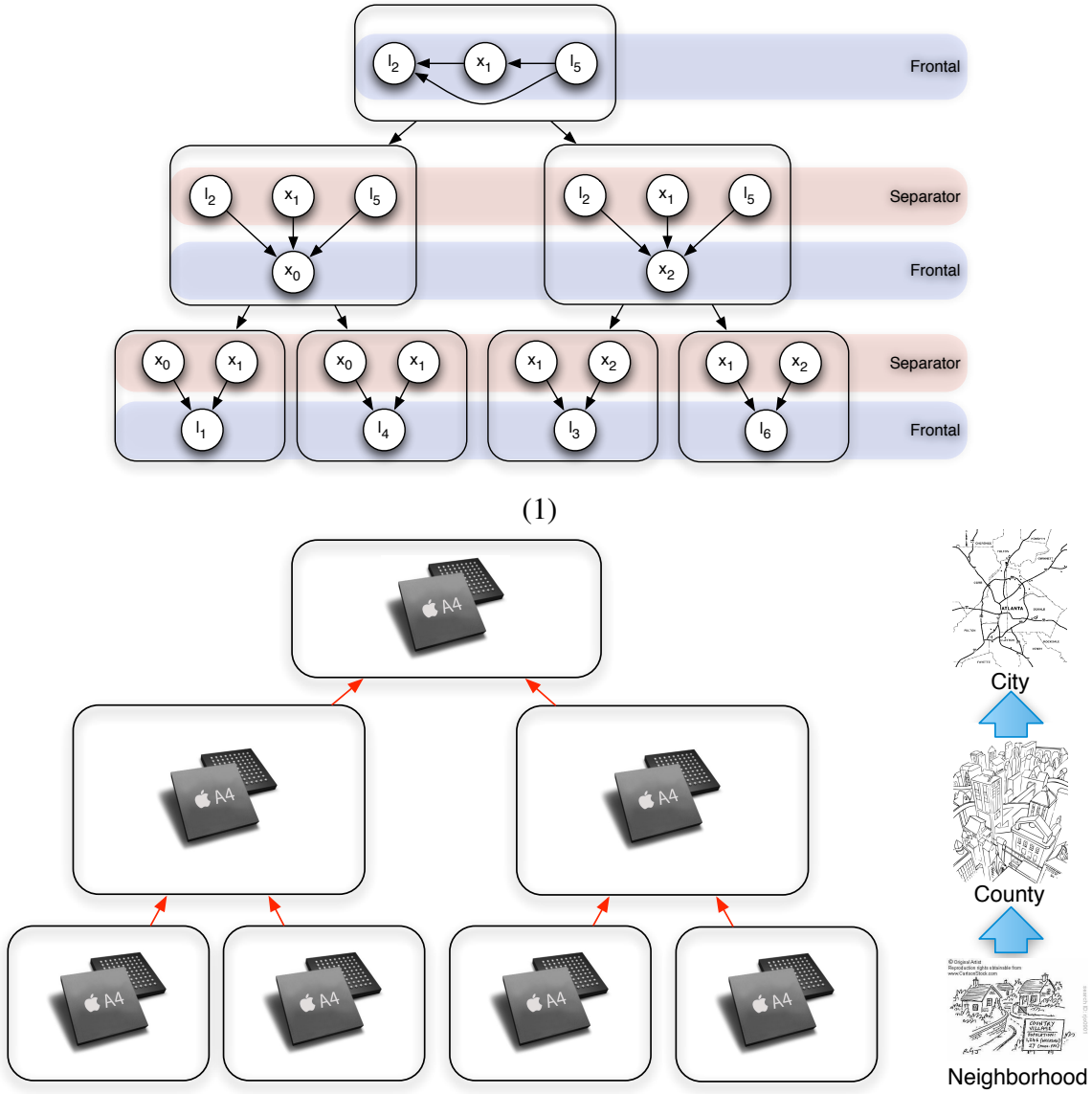
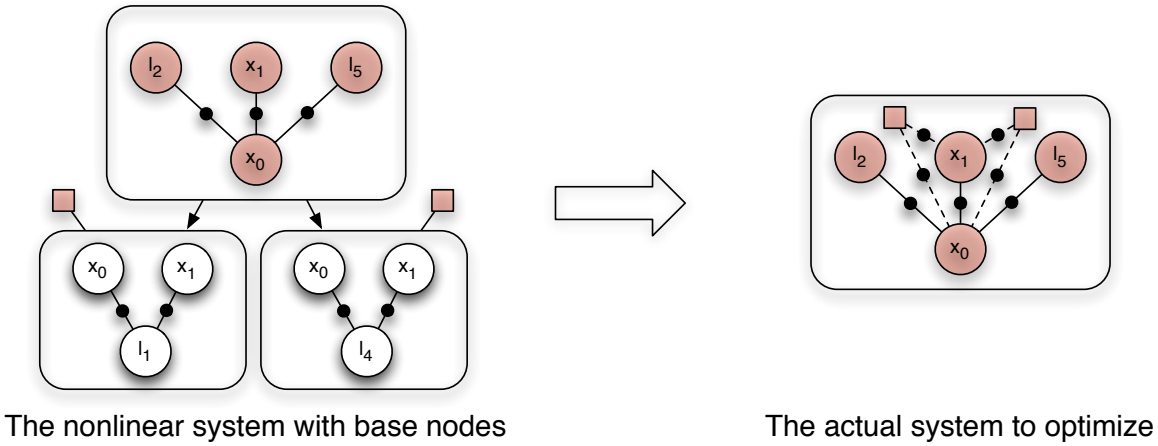
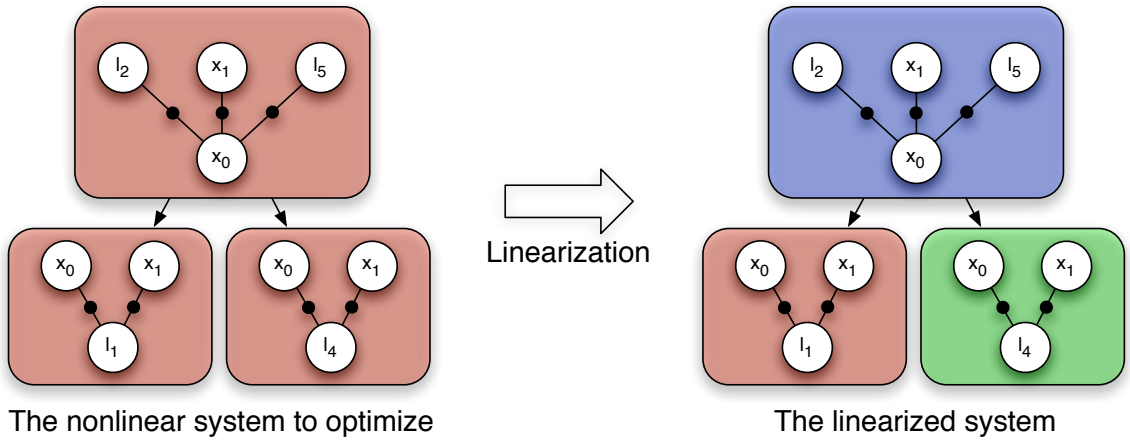


Figure 36: **The linear system solving can be done out-of-core using the cluster tree data structure.** *(1) the linear system represented by the corresponding cluster tree. It can be solved by the elimination from bottom up and the back-substitution from top down, as illustrated in Figure 13; (2) Each cluster can be assigned one core that handles the local computation, and only limited data needs to be transferred from the child clusters to their parents, indicated by the red arrows. In a realistic deployment, each cluster may represent neighborhoods, counties, and cities as the level goes up.*



(1)



(2)

Figure 37: **The nonlinear optimization can be done out-of-core for both steps illustrated in Figure 16.** *(1) The submap alignment with base nodes only involves the variables in the parent cluster as well as two base nodes; (2) The full nonlinear optimization in the subtree is done by iteratively solving the linear system at the latest estimate, which has been shown in Figure 36 hence is out-of-core as well.*

out-of-core manner. First, submaps make it possible to distribute most of the work across multiple computational resources and increases the scalability in terms of both CPU time and memory. Second, a lot of real data has different levels of noise from one portion to another, and the divide-and-conquer scheme can be easily used to allocate the computation resource smartly. In contrast, non-submap based approaches spend the resource evenly which leads to suboptimal workload scheduling.

The linear solving using Tectonic SAM is inherently parallel and out-of-core. As illustrated in Figure 36, the linear system solving can be handled by a hierarchical set of CPU cores, and the computation in the clusters within the same level can be processed in parallel. As the computation in each core only involves the measurements inside the corresponding submap, only those measurements need to be loaded into the memory of the cluster core. Hence, this hierarchically structured computation can be naturally scaled to handle real-world data from neighborhoods, counties, and even cities in the large-scale mapping applications.

The nonlinear optimization by TSAM features the same properties as well, depicted in Figure 37 . As we discussed in Section 3, the nonlinear optimization for each subtree consists of two steps. In the first step, the root cluster of the current subtree is optimized together with the base nodes of its child clusters. This computation can be done in parallel for all the subtrees in the same level, as they are statistically independent given their common ancestor. As only the base nodes are involved in additional to the local measurements of the root cluster, the memory requirement is the same as solving a linear system in a single cluster. In the second step, the entire subtree is iteratively optimized by solving the linear system at the latest estimate. Such a process is identical to the linear case hence is parallel and out-of-core as well.

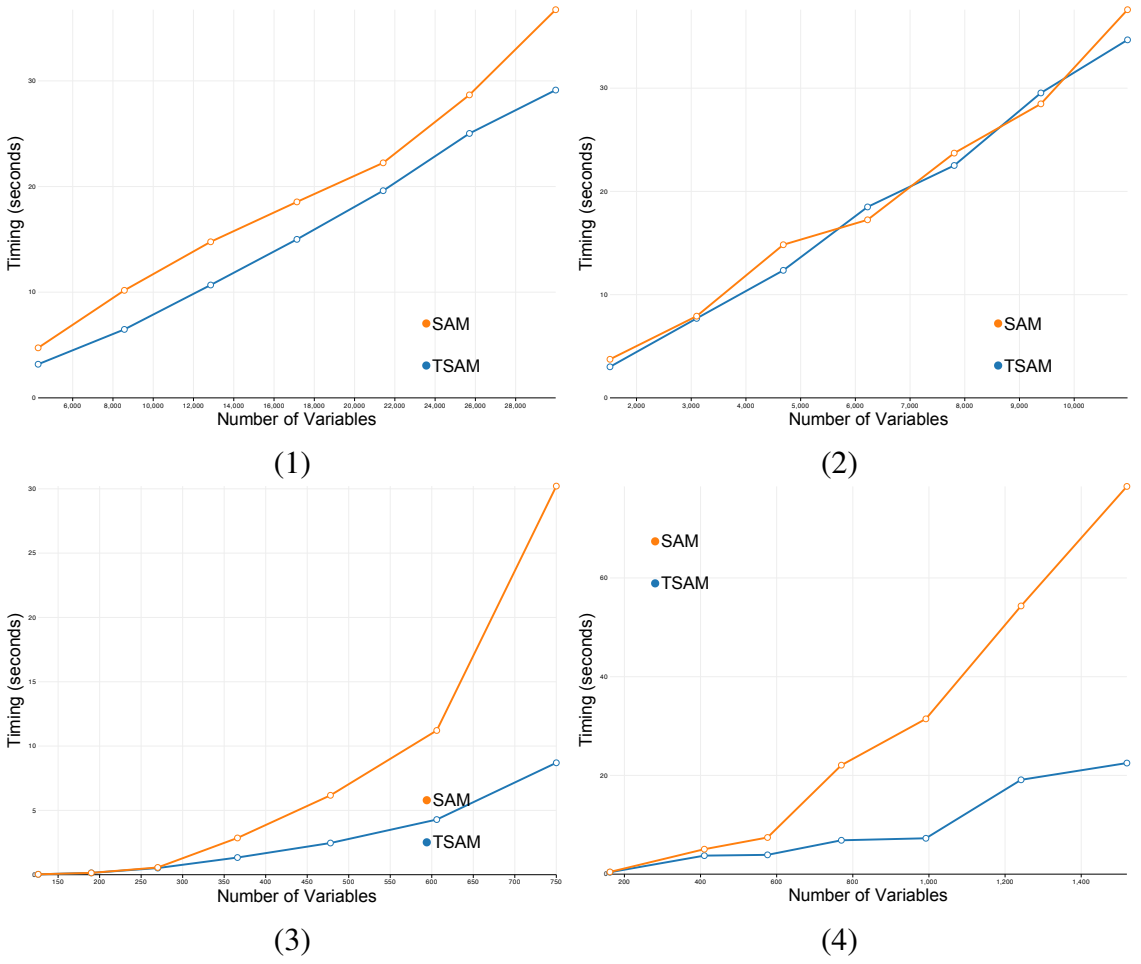


Figure 38: **The number of variables versus the timing when applying SAM and TSAM to the four simulations**: *(1) straight; (2) loop; (3) spiral, and (4) block-world data sets.*

3.4.2 Experimental Results for Scalability Tests

In this section, we evaluate the scalability of the TSAM algorithm when dealing with large-scale data-sets. We first use our in-house simulation data to evaluate the performance change when the size of the data-sets increases using the SAM as the baseline. Then we compare TSAM's performance with other state-of-the-art approaches on a large-scale simulation data-set.

First we evaluate the efficiency of the TSAM algorithm when the size of simulation data increases. For all the four simulations presented in Chapter V, we summarize the results by averaging the results from 20 runs, and in each run the input data was created by adding

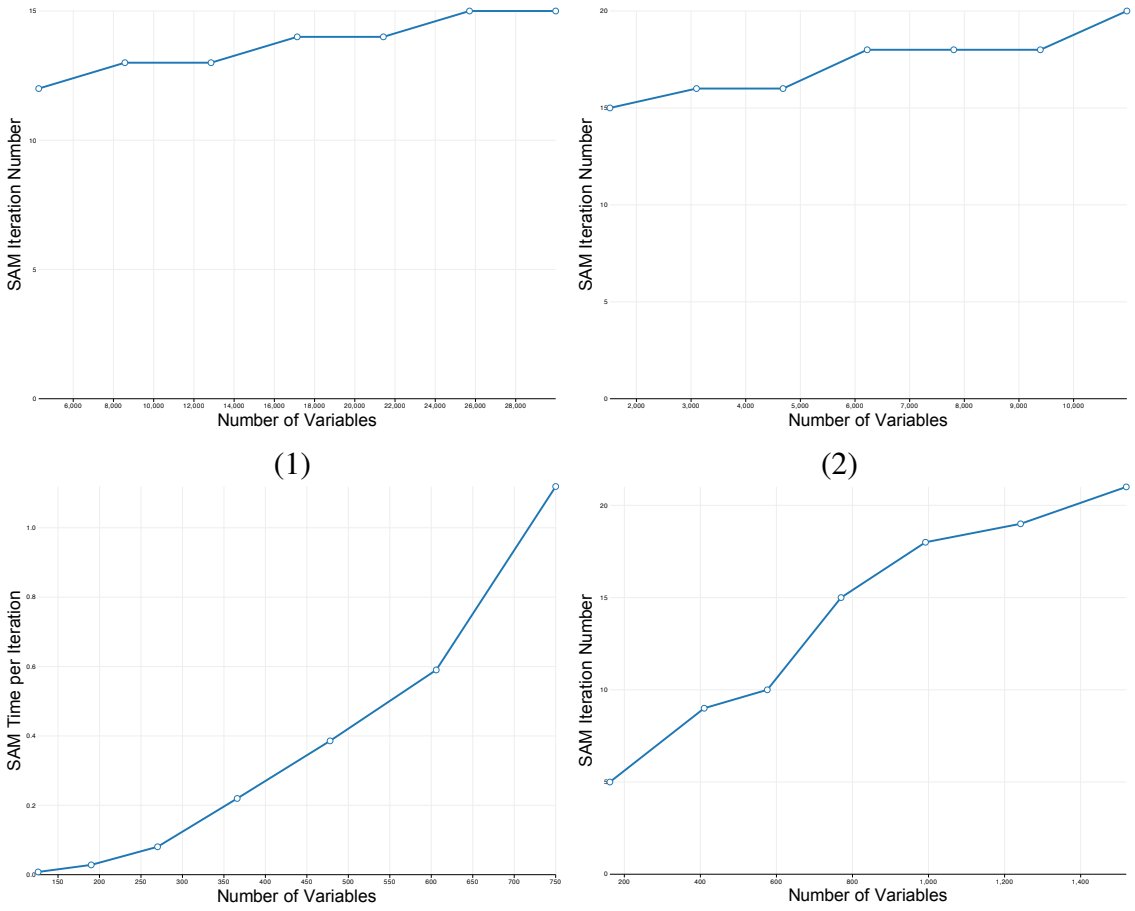


Figure 39: **The number of variables versus the total number of iterations when applying SAM to the four simulations:** *(1) straight; (2) loop; (3) spiral, and (4) block-world data sets.*

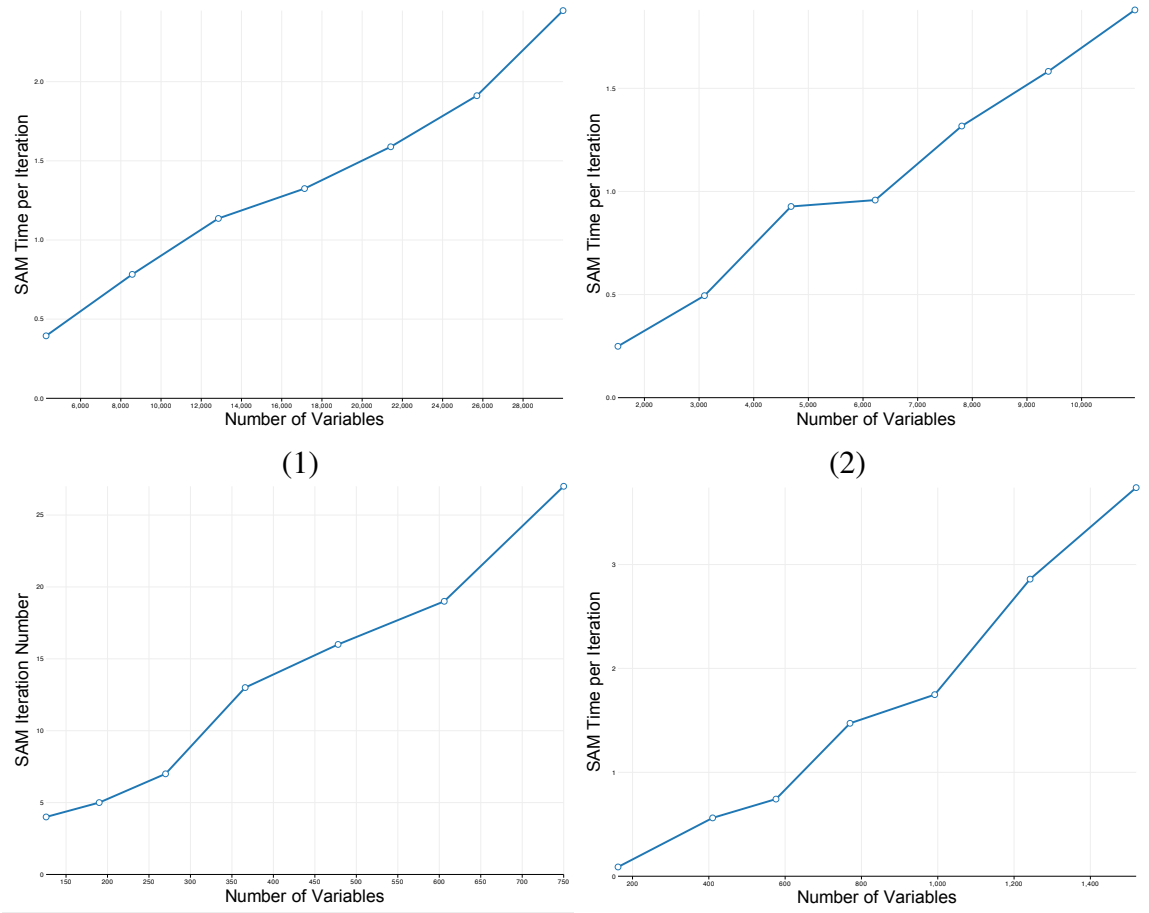


Figure 40: **The number of variables versus the time per iteration when applying TSAM to the four simulations:** *(1) straight; (2) loop; (3) spiral, and (4) block-world data sets.*

Gaussian noise with standard deviation 0.1 to the ground-truth data. For the reference, we again use the SAM [19] implementation for comparison in all the experiments.

TSAM shows more advantages on the efficiency compared to SAM when dealing with complicated scenarios where many loops exist. In particular, the robots in the straight data-set and loop data-set are exploring new environments for most of time, and the resulting simulations also have significantly fewer constraints per variable than spiral and block-world data-sets. Hence, SAM and TSAM run more efficiently in the first two simulations, as shown in Figure 38. When the problem gets more complicated, i.e. the robots frequently see some previously visited landmarks, there are considerably more loopy constraints. The performance difference can be clearly spotted in the last two simulations. Overall, the cost grows linearly for simpler scenarios, and it grows almost quadratically in the spiral-moving and block-world-moving simulations.

The different behaviors are mainly due to the variant on the number of iterations when processing four data-sets. In the straight and loop simulation, as the number of variables increases, the iteration number does not change as dramatically as the time per iteration. However, for more complicated scenarios (spiral and block-world data sets), we observe the convergence apparently became much more difficult, as illustrated in Figure 39.

On the other hand, the time per iteration has more consistent behavior across all the simulations than the iteration number (Figure 40), indicating that the number of variables are its single dominant factor. Thanks for SAM's good elimination ordering, the time per iteration is not influenced by the number of constraints very much.

The public data-set we used to verify the scalability of our algorithm is W-10000 included in the HOG-Man release [40], a SLAM algorithm that also exploits the hierarchical idea. The final map produced by TSAM is illustrated in Figure 41. We found that the HOG-Man code ran a little faster in our experiments than what was reported in [40] due to the hardware difference. Overall, TSAM is 1.4 times faster than HOG-Man (Figure 42), and the residual of HOG-Man after the convergence is 3% higher than our algorithm.

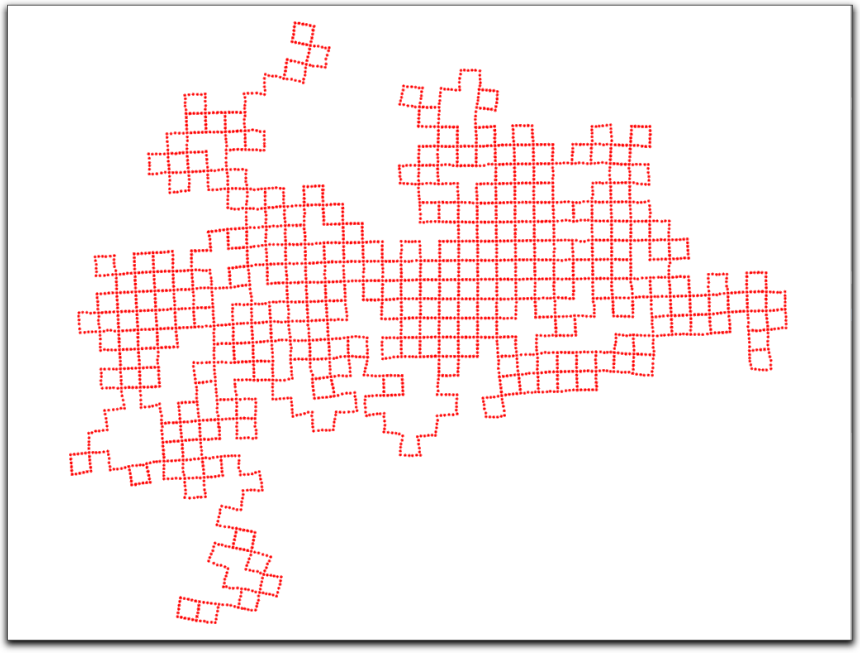


Figure 41: **The optimized map of W-10000 data-set by TSAM.**

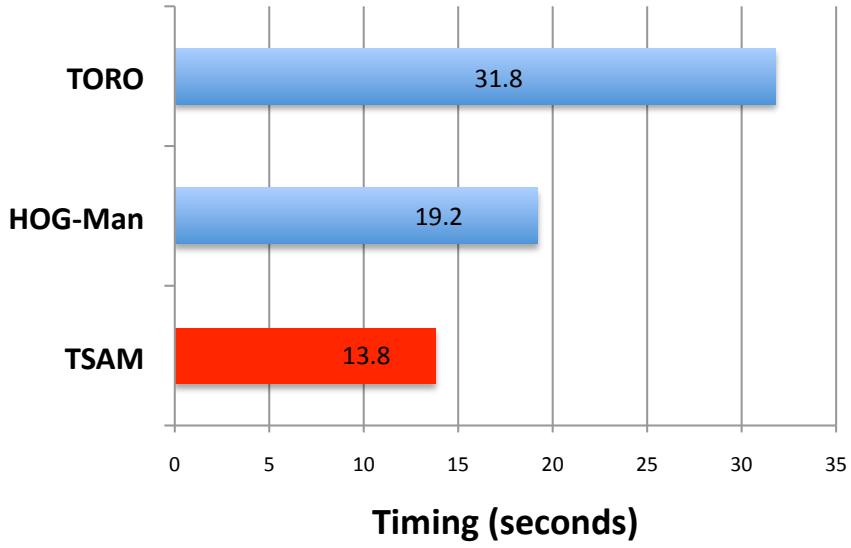


Figure 42: **The comparison of timing results on W-10000 between TORO, HOG-Man, and TSAM.**

We also compared the results with another state-of-the-art batch algorithm, TORO [43]. We ran the batch version of TORO for 300 iterations, and its final residual is 6% higher than that from TSAM. Our algorithm not only produced better maps but also converged 2.3 times faster than TORO as shown in Figure 42.

3.5 Summary

So far, we have introduced the four appealing properties of the TSAM algorithm and also demonstrated that TSAM is fast, exact, robust, and scalable using both simulation data and real data. Overall, when dealing with large-scale SLAM problems, TSAM can be a very useful divide-and-conquer method because of its powerful tree representation for partitioning the original problem into sub-problems. In fact, all the four properties are achieved more or less because of the tree representation.

Chapter IV

TSAM FOR SFM PROBLEMS

In the previous chapters, we have shown that both the 2D SLAM problem and the pose SLAM problem can both be represented by the factor graph and can be solved by Tectonic SAM after recursively partitioning the factor graph into a cluster tree using the nested dissection algorithm. In this chapter, we will extend our algorithm to tackle the *visual SLAM* or *Structure from Motion* (SfM) problem [96].

We propose a principled way to partition the SfM problem, which exploits the bipartite structure of the SfM visibility graph and convert it to a simplified camera *hypergraph*. It is shown that vertex separators composed of only 3D points can be located from the hypergraph, and non-singularity can be strictly enforced by imposing a graph refinement step after partitioning.

Our algorithm is not only out-of-core but also naturally alleviates the initialization issue of bundle adjustment in SfM problems [95, 23]. We employ the same bottom-up optimization using the submap tree obtained by recursive partitioning, as we discussed earlier. The optimization over different subtrees at the same level can be carried out in parallel. The optimized 3D submaps are aligned to one another after passing up the tree to their common ancestor, and as a consequence we never need to generate the initialization for a large problem.

4.1 Problem Formulation

In SfM [96], we infer the structure of the scene and the motion of the camera by using the correspondences between features from different views. In particular, certain types of features [68] (points, lines, and so forth) are first extracted and matched across all the image pairs. Then the camera parameters and feature locations are optimized to minimize a cost

function, such as the 2D projection errors.

The non-linear minimization of the projection errors is referred to as *bundle adjustment* in the literature [95], in which we jointly estimate the optimal 3D structure, as well as the camera parameters by minimizing a least-squares cost function. Typically, the measurement function $h_k(.)$ is non-linear, and one assumes a normally distributed measurement noise with associated covariance matrix Σ_k, leading to

$$\sum_{k=1}^{K} \left\| h_k(C_{i_k}, P_{j_k}) - z_k \right\|_{\Sigma_k}^2 \tag{10}$$

Above, $C_i (i \in 1...M)$ represents the intrinsic and extrinsic camera calibrations, $P_j (j \in 1...N)$ represents the 3D structure, and $z_k (k \in 1 \ldots K)$ represents the 2D measurement of the point P_{j_k} in camera C_{i_k}. The notation $\|.\|_{\Sigma}^2$ stands for the squared Mahalanobis distance with covariance matrix Σ.

4.1.1 SfM as a Bipartite Visibility Graph

The SfM problem can be represented using a *visibility graph*, as shown at the top of Figure 43. The visibility graph for the SfM problem is the bipartite graph $G_{\text{SfM}} = (C, P, E)$ where the sets of cameras C and points P appear as vertices, and there is an edge e_{ij} corresponding to every measurement z_k, indicating that point P_j is visible in camera C_i. Note that dividing the original SfM problem is equivalent to partitioning the corresponding visibility graph. Generally speaking, we desire that the sub-problems have similar sizes, and each sub-problem is also self-contained, i.e. non-singular.

4.2 Tectonic SAM for SfM

Within the same Tectonic SAM framework, the SfM approach we introduce here includes three major parts:

1. A hierarchical partitioning based on hypergraphs.

2. A refinement step that deals with degeneracies.

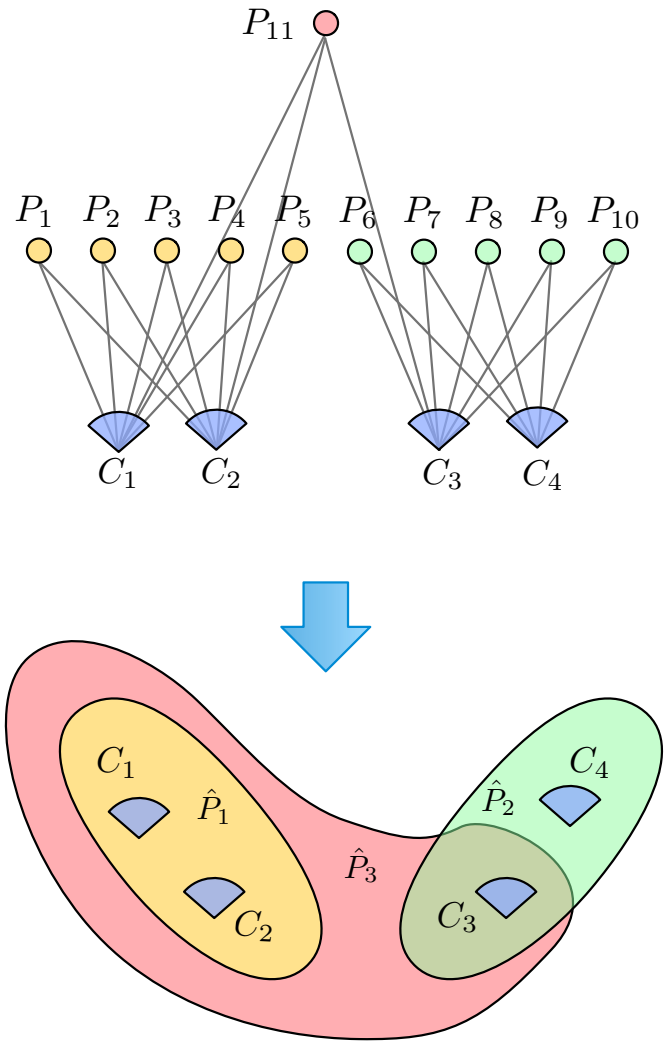


Figure 43: **The visibility graph of an exemplar SfM problem on the left is converted to the corresponding hypergraph representation on the right**. *The cameras are shown in blue, and the 3D points are shown in yellow. Each edge in the left graph indicate that a 3D point is visible from a certain camera. The contours in the right graph represent the hyperedges in the hypergraph, and each contour connects multiple cameras. Note that the colors of 3D points in the visibility graph share the same colors as the corresponding hyperedges in the hypergraph.*

3. A bottom up optimization step that merges submaps.

Compared to the approach we introduced earlier for the 2D SLAM problem, we employ a new graph representation which best fits the SfM problems and also take special care of the degeneracy problem arisen in the SfM problem.

4.2.1 Hierarchical Partitioning for SfM

Our algorithm works by finding a small edge separator in the *camera hypergraph*, which corresponds to a vertex separator in the original visibility graph consisting solely of 3D points. A *hypergraph* $\mathscr{H} = (\mathscr{X}, \mathscr{E})$ is composed of a set of vertices $\mathscr{X}$ and a set of hyperedges $\mathscr{E}$, where each hyperedge connects multiple vertices. We define the camera hypergraph $\mathscr{H}_{\text{cam}} = (C, P)$ of an SfM problem as the hypergraph obtained by treating the cameras C as vertices $\mathscr{X}$ and the 3D points P as hyperedges $\mathscr{E}$.

To reduce the complexity of hypergraph $\mathscr{H}_{\text{cam}}$ and speed up later graph operations, we introduce *compressed edges.* In SfM problems, it is typical that two or more images perceive the same cloud of 3D points, e.g. points on a building facade. Those 3D points correspond to the edges in $\mathscr{H}_{\text{cam}}$ that connect to the same camera vertices. Hence, it is straight-forward to introduce compressed edges and generate a much simplified graph $\hat{\mathscr{H}}_{\text{cam}} = (\mathscr{X}, \hat{\mathscr{E}})$. For example, points P_1 to P_5 at the top of Figure 43.(1) all connect to C_1 and C_2, so they will appear as a single edge $\hat{P}_1$ in $\hat{\mathscr{H}}_{\text{cam}}$, as shown at the bottom of Figure 43.(2). We also specify the weights of the resulting hyperedges using the number of 3D points each compressed edge represents, e.g. $w_{\hat{P}_1} = 5$.

We partition the hypergraph by finding a small edge separator $\hat{\mathscr{E}}_S$ (defined below). This hypergraph partitioning is done recursively, which leads to a tree structure as shown in Figure 44. The recursive partitioning stops when the size of the current submap is smaller than a certain threshold (5000 for all our experiments). Note that the sizes of the separator are typically very small (details will be described in the result section) In fact, although there are no theoretical guarantees for the separator size in SfM problems, we found we

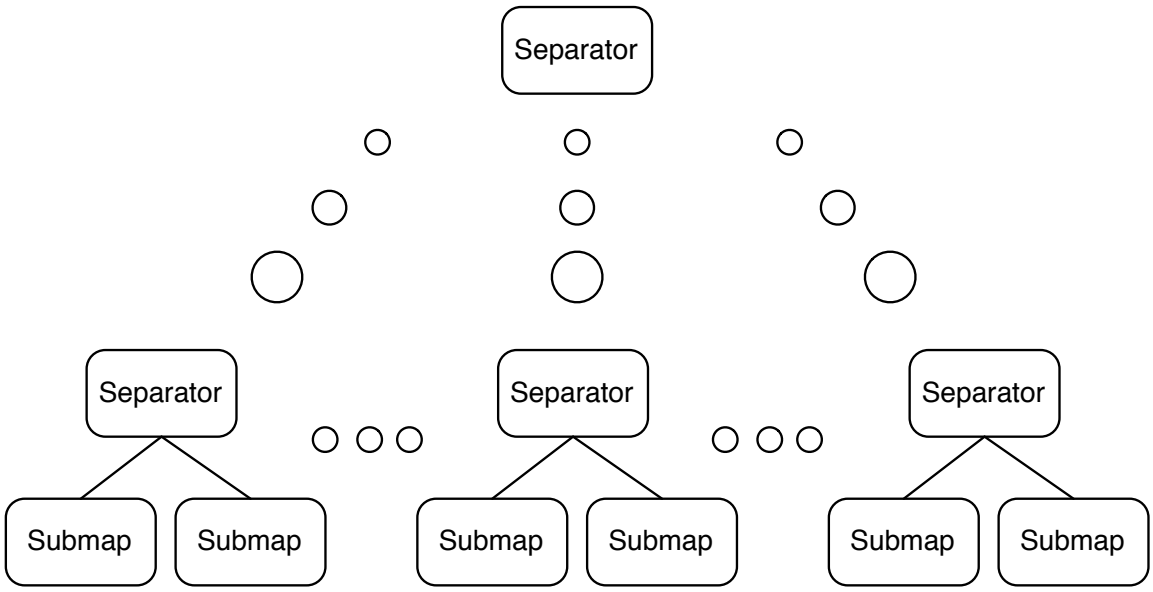


Figure 44: **By partitioning the hypergraphs and finding vertex separators in the visibility graph, the original SfM problem can be partitioned recursively**. *The resulting tree structure has submaps on its leaves and has separators along the path from the leaves to the root. Note that two subtrees are independent given their common ancestor on the tree.*

can in practice always find good separators by exploiting the underlying structure of SfM.

An *edge-separator* $\hat{\mathscr{E}}_S$ of a hypergraph $\hat{\mathscr{H}}_{\text{cam}}$ is a subset of its edges $\hat{\mathscr{E}}$ that disconnects $\hat{\mathscr{H}}_{\text{cam}}$ into two or more separate connected components $\hat{\mathscr{H}}_A = (C_A, \hat{P}_A), \hat{\mathscr{H}}_B = (C_B, \hat{P}_B) \ldots$. Because of the definition of camera hypergraph $\hat{\mathscr{H}}_{\text{cam}}$, any edge separator $\hat{\mathscr{E}}_S$ in $\hat{\mathscr{H}}_{\text{cam}}$ automatically corresponds to a subset P_S of the 3D points in the original visibility graph G_{SfM}. For example, in Figure 45, edge separator $\hat{\mathscr{E}}_S = \hat{P}_3$ corresponds to the 3D point set $P_S = \{P_1\}$ from the visibility graph.

Theorem 1. *If $\hat{\mathscr{E}}_S$ is the edge separator of hypergraph $\hat{\mathscr{H}}_{\text{cam}}$,* and *$P_S$ is the set of its corresponding 3D points in the visibility graph G_{SfM}, we have that P_S is the vertex separator of G_{SfM}, disconnecting the visibility graph $G_{\text{SfM}} = (C, P, E)$ in two or more components $(C_A, P_A, E_A), (C_B, P_B, E_B), \ldots$*

Proof. It is easy to see that the cameras in C_A and C_B in G_{SfM} are not connected, because G_{SfM} is bipartite. But moreover, no point in P_A is visible from any camera in C_B. To see this, assume there is point $P_1 \in P_A$ connected to a camera $C_1 \in C_B$. Because (C_A, P_A, E_A) is

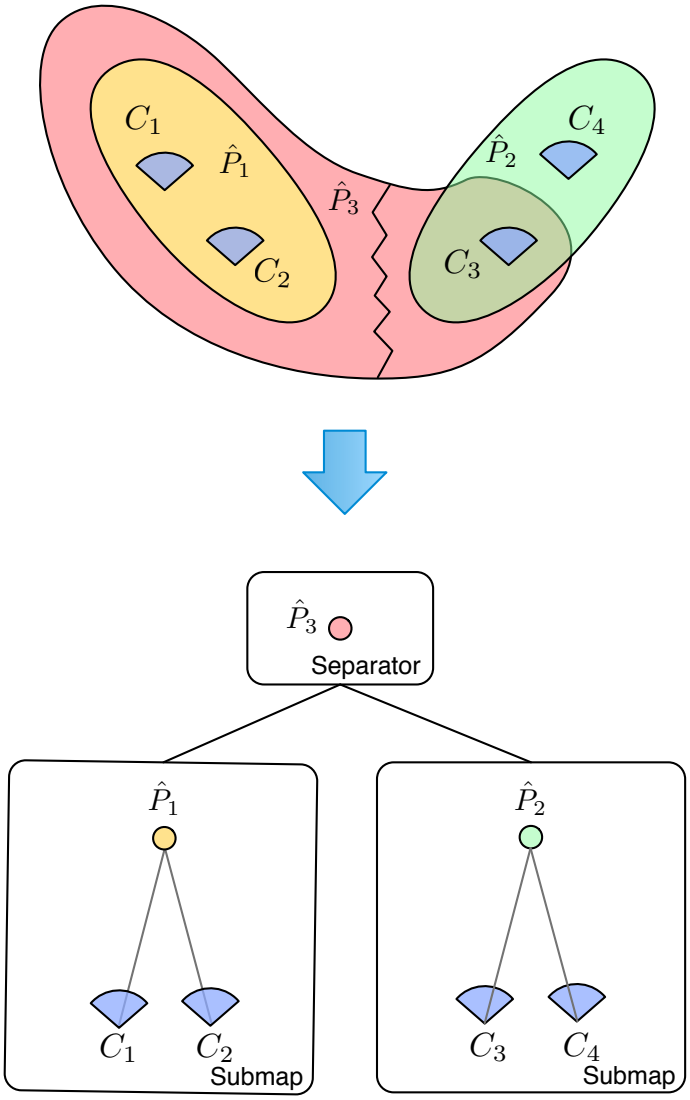


Figure 45: **Partitioning a hypergraph for a SfM problem (top) using an edge separator is equivalent to finding a vertex separator composed solely of 3D points in the original visibility graph (bottom).** *The edges in the hypergraph are weighted according to the number of 3D points they correspond to, and $\hat{P}_3$ is chosen as the edge separator here because it has the smallest weights. The two resulting submaps are independent to each other given their vertex separator and can be optimized in an out-of-core manner.*

a connected component (by definition), P_1 is also connected to some camera, say C_2 in C_A. But then P_1 must be in the edge separator $\hat{\mathscr{E}}_S = P_S$, and not in P_A, which is a contradiction. QED.

4.2.2 Degeneracy Issues and Graph Refinement

To be able to optimize the partitioned submaps separately, we need to make sure that each submap is fully constrained. That is to say, each landmark is visible in at least n cameras, and each camera perceives at least m landmarks. For example, in a SfM problem where camera calibrations are known, we have $m = 5$ and $n = 2$. Combining both requirements, we call such a partitioning a constrained (m,n)-*cut*.

In the partitioning algorithm we introduced above, we explicitly choose a vertex separator P_S consisting solely of 3D points. The main reason for doing this is that it prevents the 3D points in the resulting submaps $(C_A,P_A,E_A),(C_B,P_B,E_B),\ldots$ from becoming singular. In SfM problems, the 3D points are typically only visible to a small number of cameras and can easily become under-constrained if one or more of their cameras end up in the vertex separator after the partitioning. Hence, to generate a valid (m,n)-cut, we try to use only 3D point vertices as the separator for G_{SfM}. In this case, only cameras in the submaps "lose" constraints if their observable 3D points are part of the separator. As each camera usually sees a lot of 3D points, losing some of those constraints because of vertex separator P_S usually does not make the camera singular. Also note that *all* the 3D points in the resulting submaps after partitioning remain fully constrained, as their connections to the neighboring cameras stay same.

We also employ a partition refinement step to enforce non-singularity and handle the rare case that some cameras become singular after we split the graph using a certain vertex separator, as shown in Figure 46. In graph theory, graph refinement refers to the local improvement approaches such as the widely-used KL refinement by Kernighan and Lin [54]

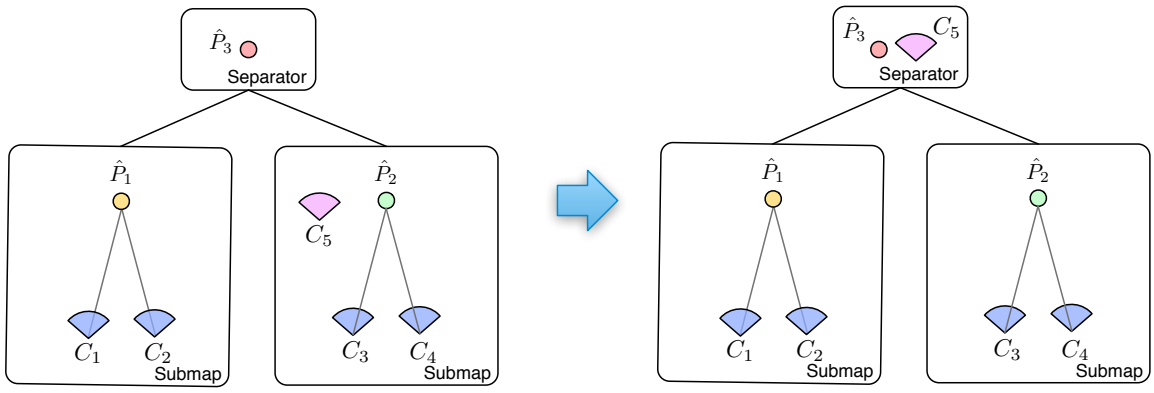


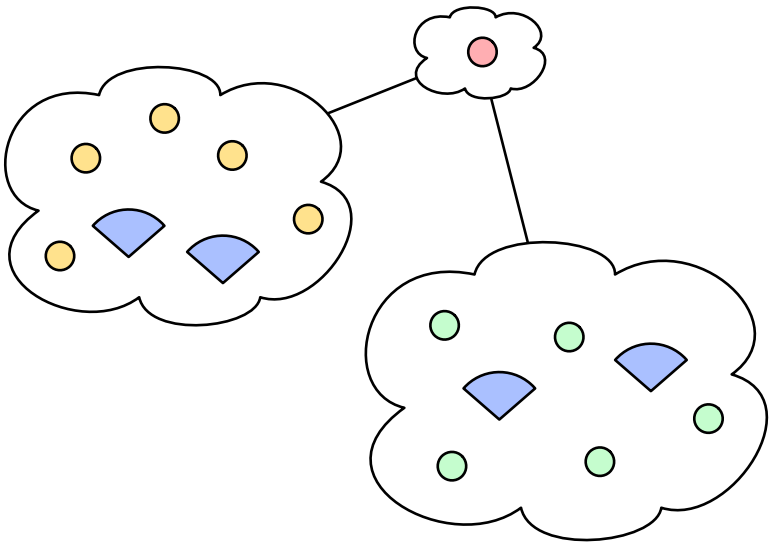
Figure 46: **The partition refinement step that enforces the non-singularity of each variable.** In each iteration, all the singular variables in the current submaps are moved to the separator. The refinement step finishes when there is no singular variable found in any submap.

and the algorithm by Fiduccia and Mattheyses [26]. In this work, given hypergraph partitioning results, we locate the under-constrained cameras in submaps $(C_A, P_A, E_A), (C_B, P_B, E_B), \ldots$ and put them into the separator P_S. Up to this point, we also need to check all the affected 3D points in $P_A, P_B, \ldots$ and put the under-constrained ones to P_S as well. We iterate over cameras and 3D points until all the vertices in the submaps are fully constrained. For all the data sets we tested later in this Chapter, it took at most two iterations, which makes graph refinement very efficient.

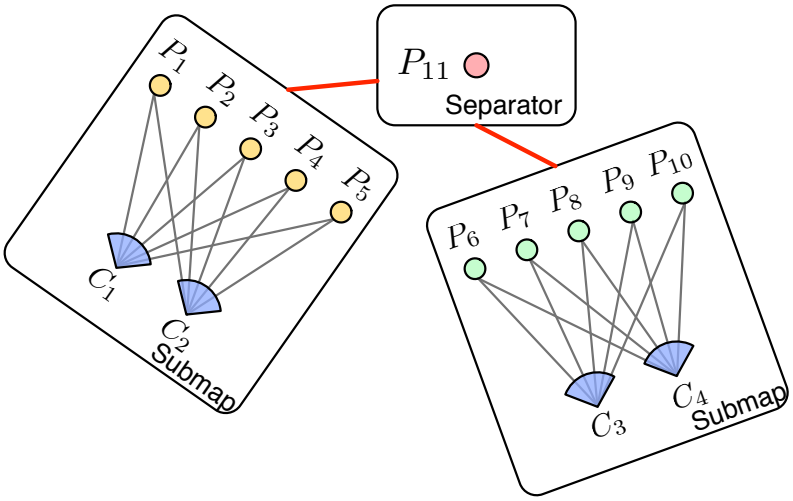
4.2.3 Bottom-up Optimization

Given the tree structure after recursive partitioning, we employ a bottom-up process to optimize and merge all the submaps into a final global reconstruction. The process is inherently recursive (illustrated in Figure 47): for each subtree, the separator waits for its children to complete their own optimization. All the optimized child submaps are then aligned with each other as *rigid* maps. At last, all the child submaps as well as their separators are optimized together in the unified coordinate system. Such a process is carried out for each separator, and the entire SfM problem is solved in a bottom-up fashion.

The bottom-up process can be done efficiently in an out-of-core manner. Note that the multi-level tree structure induces more submaps with smaller sizes compared to the

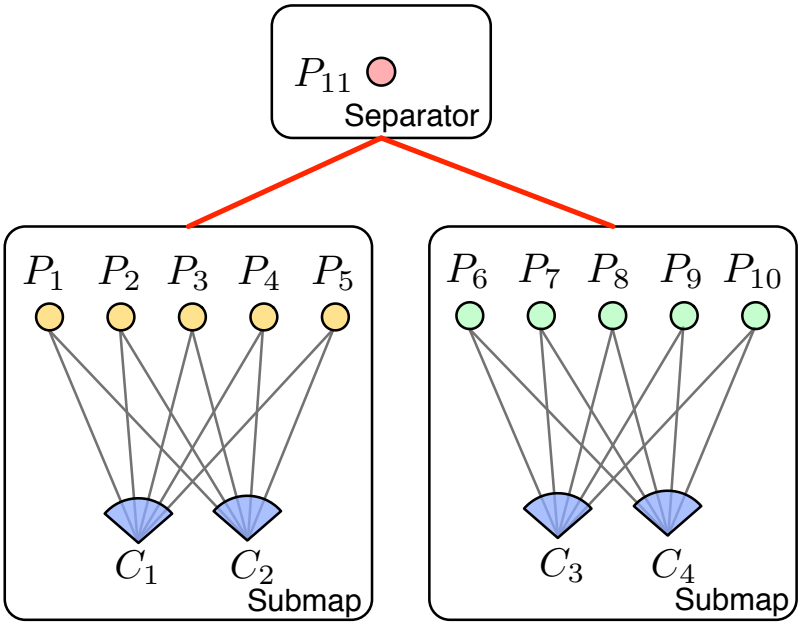


(1) Wait for the child subtrees/submaps to be optimized

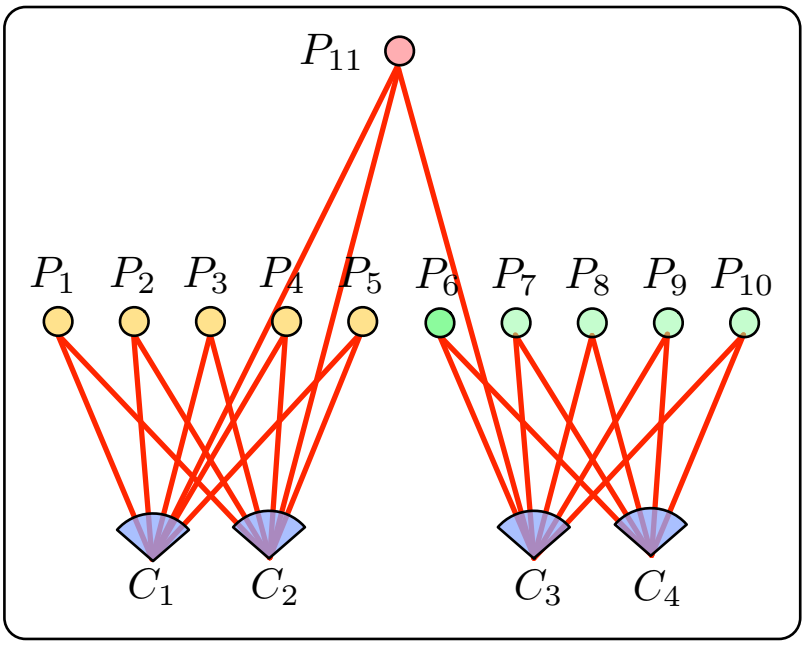


(2) Submaps locally optimized but misaligned with each other

Figure 47: **The bottom-up optimization is carried out recursively.** *The red edges indicate the constraints used in each optimization.*



(3) Align submaps using base nodes



(4) Merge the submaps and smooth the map globally

Figure 48: **Continuation of the bottom-up optimization in Figure 47.** *The red edges indicate the constraints used in each optimization.*

single-level submap based approaches such as [74], and this enables us to distribute the computation to more cores.

The rigid alignment of submaps is achieved using base nodes, as shown in step (3) of Figure 47. Each submap is assigned a base node, which represents the 6DoF transformation between the submap and its separator. Such an alignment is fast because only several base node variables as well as a small separator are involved. In fact, the number of base nodes under the same separator is typically two or three, and the sizes of the separators are also much smaller compared to the size of the original problem (the details will be presented in the results section).

The initialization problem on each level of the tree is sidestepped by integrating the optimized submaps from the children. It has been shown that a reasonably good initialization is crucial for the final convergence of SfM problems [95, 23]. In large-scale SfM problems, the initialization is more difficult due to errors accumulated in local reconstructions, e.g. pair-wise reconstructions, greatly reduce the overall initialization quality. By splitting the original data into many smaller parts, the initialization in the global level is avoided. Instead, small maps propagate their optimized states to the parent separator, where that information is integrated together by submap alignment. In this way, more reliable initialization is achieved from the bottom up, which makes the algorithm more robust.

The bottom-up optimization proposed here is also exact. A smoothing step is executed for all the subtrees including the entire tree when reaching the root level (step (4) in Figure 48). This improves the quality of the most recent subtree estimate and supplies an initialization for the parent level, but importantly it guarantees the exactness of the current subtree: the smoothing step is the same as regular bundle adjustment with *all* variables involved. For levels other than the root level, the optimization does not need to fully converge, as its results only serve as the initialization for the next level. In our experiments, we limit the number of iterations for all subtree smoothing to seven except for the root level, where we use the same convergence criteria as we would for regular bundle adjustment.

Table 2: **The partitioning results for the five data sets**. $|P_S|$is the number of vertices in the root separator P_S, and $|G_{\text{SfM}}|$ is the total number of vertices in the original problems. The second column indicates the number of submaps after the first-level partitioning. The timing results in the last column is the total time cost of the entire recursive partitioning.

	$\lvert P_S\rvert \,/\, \lvert G_{\text{SfM}}\rvert$	Nr. Submap	Time (sec.)
Brown House	2.48%	2	0.57
Old House	1.61%	3	1.28
Grand Canal	0.99%	2	3.12
San Marco	12.5%	3	3.71
St. Peters	4.00%	2	5.10

4.3 Experimental Results

4.3.1 Hypergraph Partitioning

We demonstrate the hypergraph partitioning using five indoor and outdoor data-sets (Brown House, Old House, Grand Canal, San Marco, St. Peters) from the PhotoSynth database [90], as shown at the top of Figure 49, Figure 50, Figure 51, Figure 52, and Figure 53. After generating the camera hypergraph $\hat{\mathscr{H}}_{\text{cam}}$, we use the Metis graph partitioner [51] to find the edge separators, which are shown as the two images in those five figures. All the resulting first-level submaps are shown in other colors.

First, we evaluate how well our algorithm is able to divide SfM problems. As listed in Table 2, most of the data sets have a separator smaller than 5% of the total data size. This means that given a small part of the points and cameras (mostly points), the entire 3D world can be decoupled into two or three submaps *without* discarding any measurements from the original problem.

We investigated both indoor and outdoor scenarios, and the characteristics of the data sets vary from one to another. For the Grand Canal, there is no loop in the camera trajectory, hence the partitioning is straightforward: the set is split into two parts along the camera trajectory. In Brown House, Old House and St. Peter data sets, there are loops in the trajectory, but the cameras do not have many 3D objects in common. Hence the separator size is still very small. On the other hand, in the San Marco data set, the separator

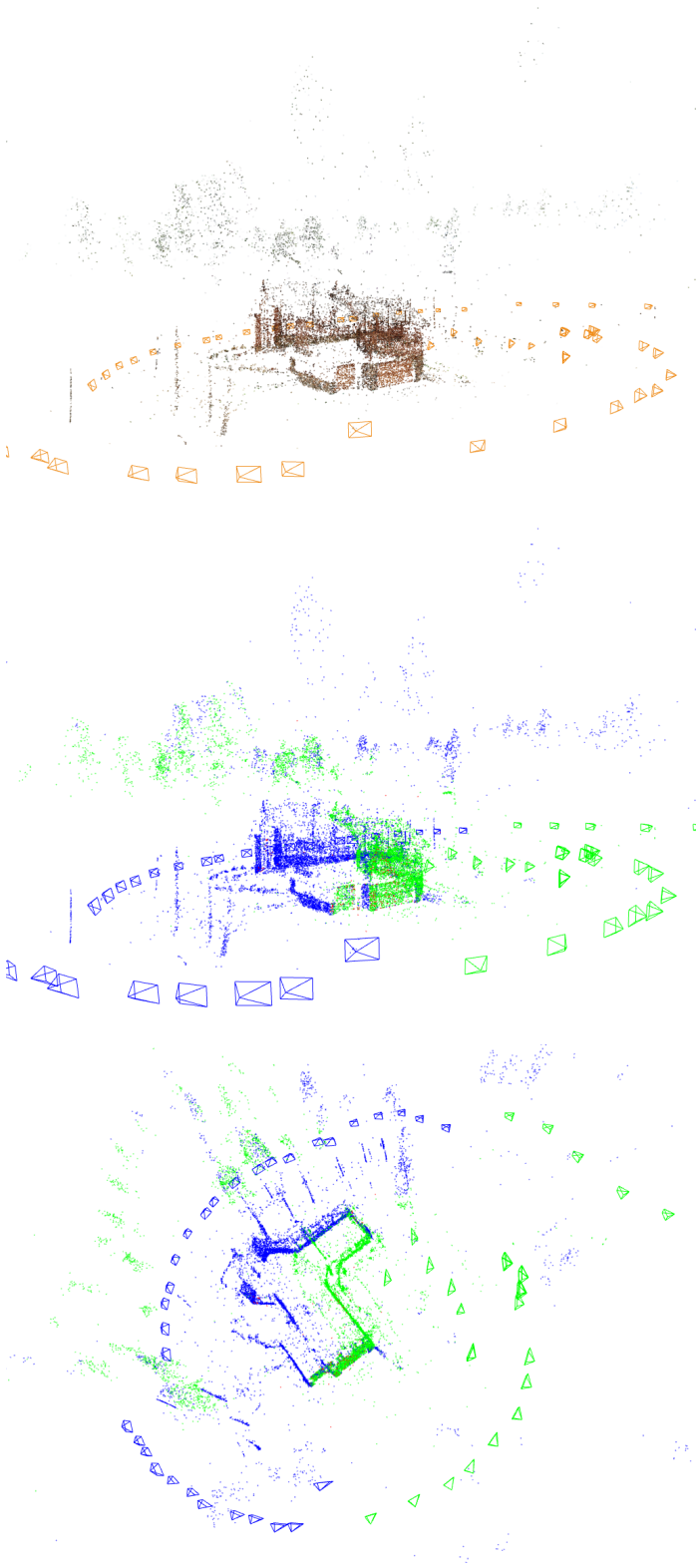


Figure 49: **The partitioned results for the Brown House data-set.** *In each of the four-image collage, the top-left image and the top-right image show the cameras and the point clouds in their original color. The bottom-left image and the bottom-right image show the front view and the top view of the root separator (labelled in red) and the first-level submaps (labelled in the other colors).*

Figure 50: **The partitioned results for the Old House data-set.** *In each of the four-image collage, the top-left image and the top-right image show the cameras and the point clouds in their original color. The bottom-left image and the bottom-right image show the front view and the top view of the root separator (labelled in red) and the first-level submaps (labelled in the other colors).*

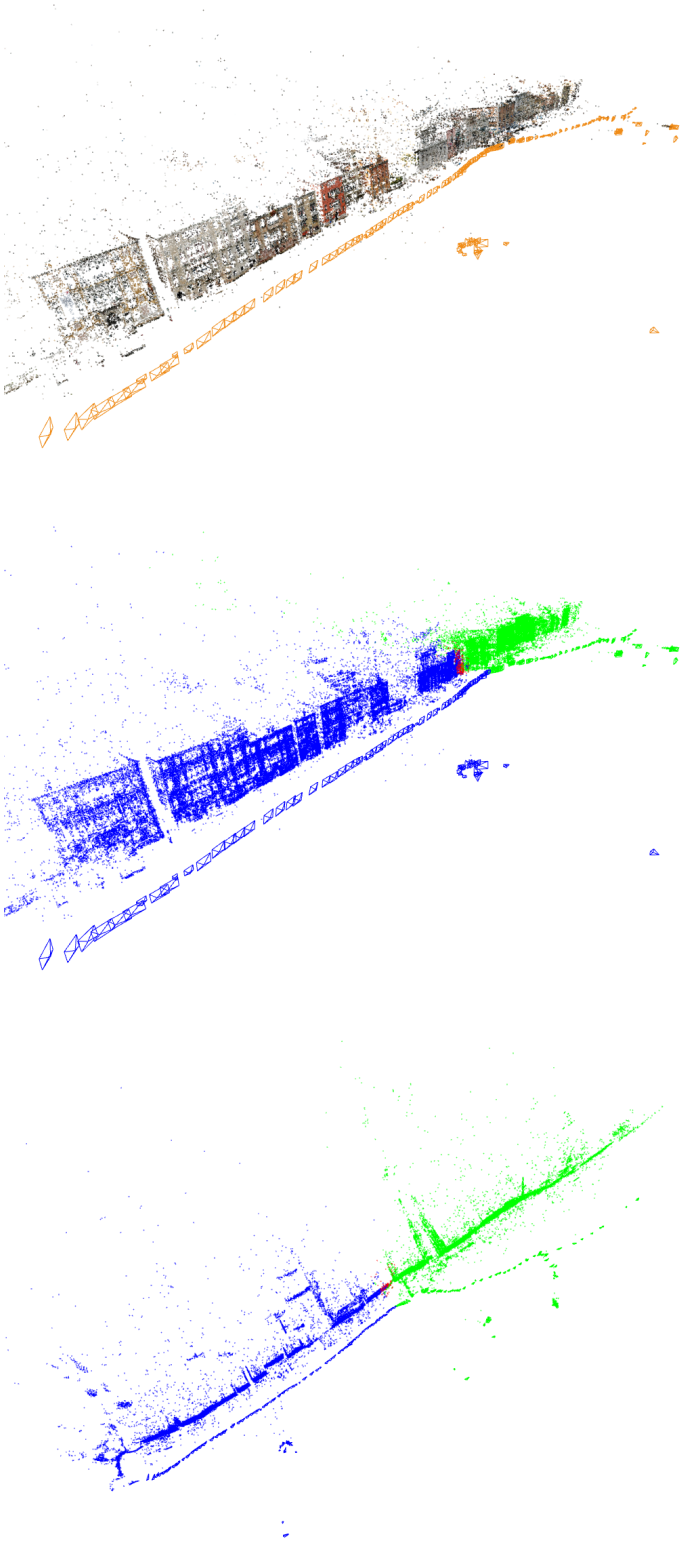


Figure 51: **The partitioned results for the Grand Canal data-set.** *In each of the four-image collage, the top-left image and the top-right image show the cameras and the point clouds in their original color. The bottom-left image and the bottom-right image show the front view and the top view of the root separator (labelled in red) and the first-level submaps (labelled in the other colors).*

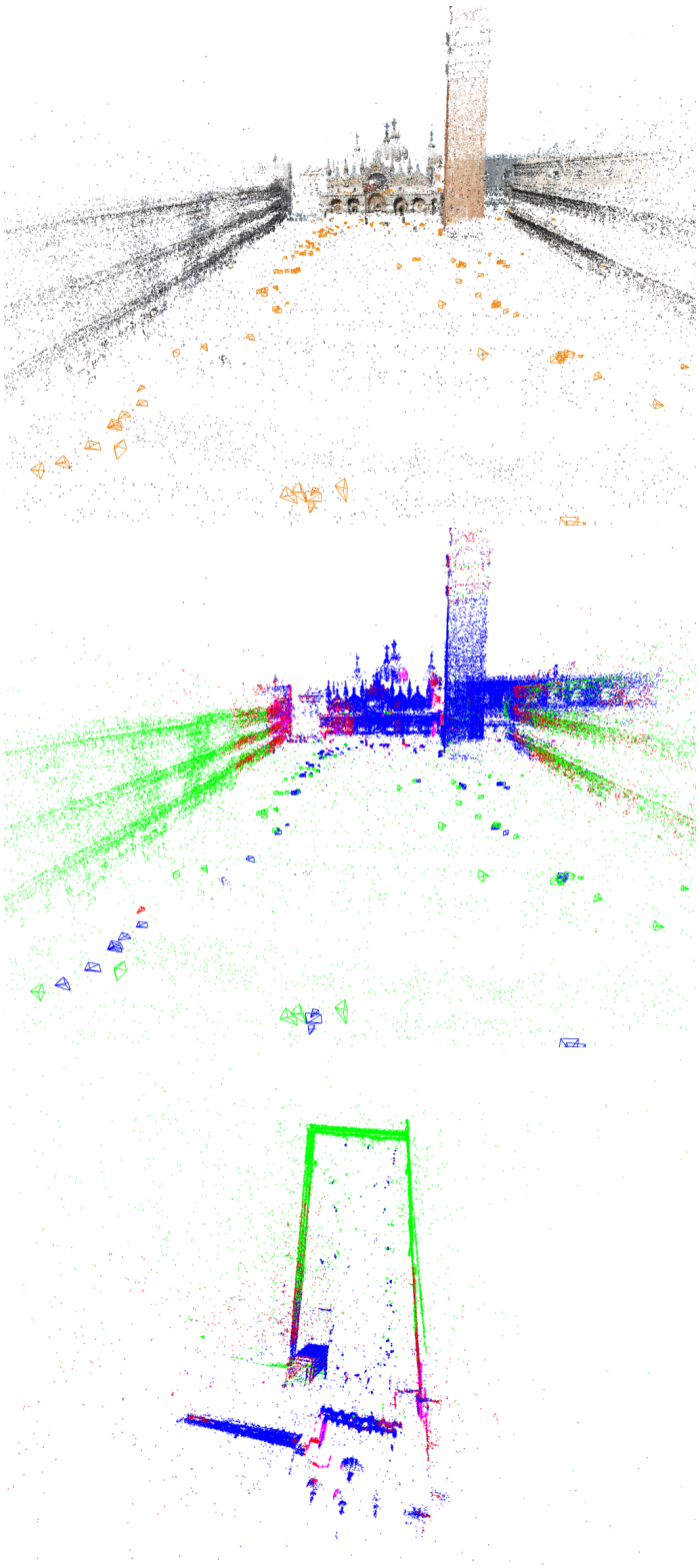


Figure 52: **The partitioned results for the San Marco data-set.** *In each of the four-image collage, the top-left image and the top-right image show the cameras and the point clouds in their original color. The bottom-left image and the bottom-right image show the front view and the top view of the root separator (labelled in red) and the first-level submaps (labelled in the other colors).*

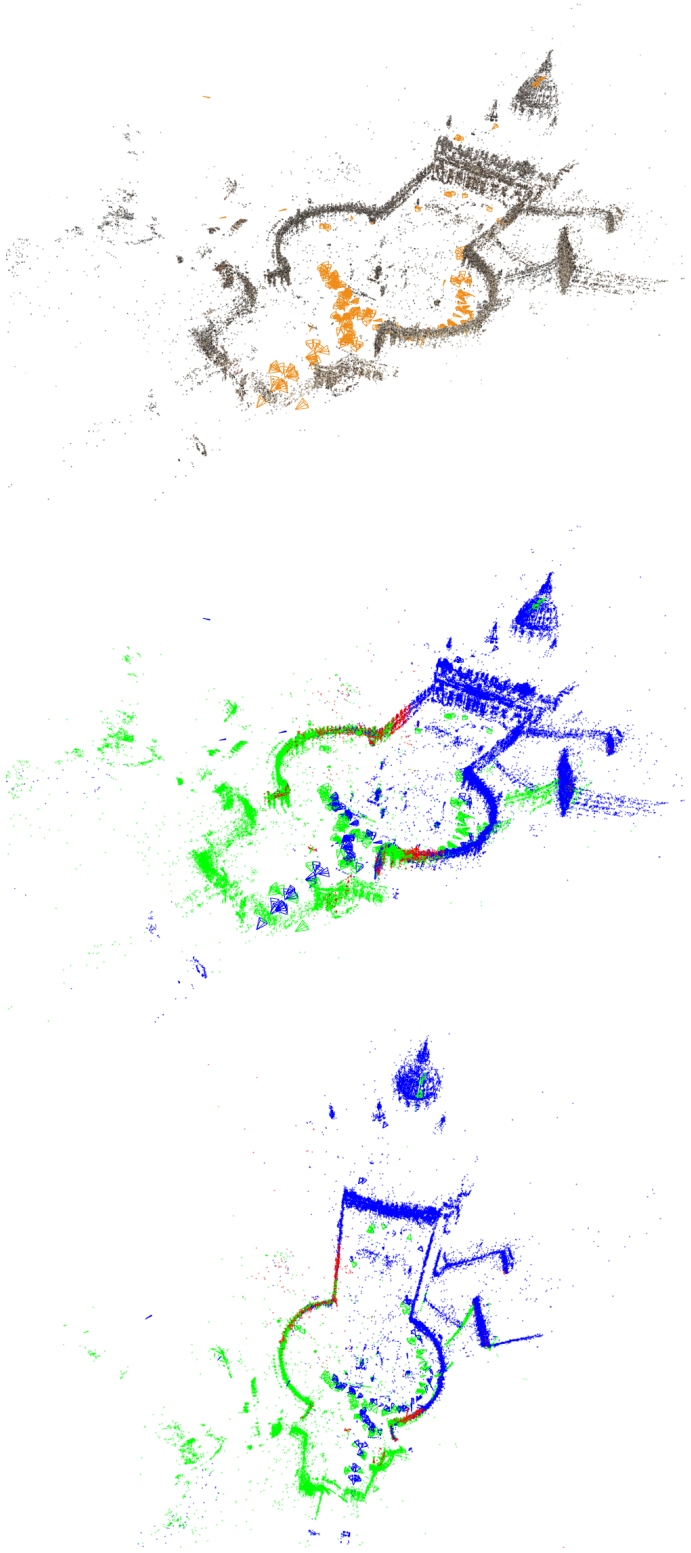


Figure 53: **The partitioned results for the St. Peter data-set.** *In each of the four-image collage, the top-left image and the top-right image show the cameras and the point clouds in their original color. The bottom-left image and the bottom-right image show the front view and the top view of the root separator (labelled in red) and the first-level submaps (labelled in the other colors).*

Table 3: **The timing results for the five data sets**. BA indicates our own implementation of regular bundle adjustment, which uses AMD ordering [4] to solve the linear systems. Note that TSAM uses exactly the same implementation as regular bundle adjustment to optimize the individual submaps.

	Cameras	BA(sec.)	TSAM(sec.)
Brown House	61	725	456
Old House	178	1279	789
Grand Canal	270	3237	1553
San Marco	237	N/A	1465
	285	N/A	1823

Table 4: **The timing results for optimizing submaps in Grand Canal data-set on each level of bottom-up optimization**. Results are averaged over all the operations that happen at the save level. In the first column, the two numbers are the average nonlinear iteration numbers and the average time per iteration for aligning the child submaps with respect to each separator. In the second column, the numbers are the corresponding iteration numbers and time per iteration for smoothing each submap. At the 5th level, submap alignment results are not available because there are no child submaps at this level.

Level	Submap Alignment	Subtree Smoothing
1st	6.0 iter., 0.48sec.	3.0 iter, 186.40 sec.
2nd	6.5 iter., 0.72 sec.	3.5 iter., 28.08sec.
3rd	8.2 iter., 1.28 sec.	4.3 iter., 7.67 sec.
4th	8.9 iter., 1.50 sec.	4.5 iter., 3.45 sec.
5th	N/A	6.3 iter., 0.78 sec.

size is bigger as most of pictures were taken along the direction of St Mark's basilica and Campanile town, hence there are far more overlapping between camera views, which leads to a bigger separator. Note that overall it is still a small portion of the original data.

The partitioning on the hypergraph is very efficient, as shown in the last column of Table 2. For all five data sets, the longest execution time is about five seconds. Compared to the total SfM timing shown in the next section, the overhead caused by partitioning is less then 1%, hence it can often be neglected in the whole SfM pipeline.

4.3.2 SfM Timing Results

In this section, we measure and compare the timing results of the regular bundle adjustment algorithm and the TSAM algorithm. All the experiments were carried out on a 2.8GHz

Intel Core 2 Duo machine with 8GB memory. We used the same feature correspondences as inputs to both algorithms, and we also made TSAM use the same implementation and settings as regular bundle adjustment when doing the nonlinear optimization for all the submaps and the final global reconstruction. Hence, the differences in the timing results are only due to TSAM's divide-and-conquer scheme.

In Table 3, we can observe that solving the decoupled problems yields a great improvement in terms of both speed and robustness. For the Grand Canal data-set and the St. Peters data set, regular sparse bundle adjustment does not converge properly because the initialization we obtained from pairwise reconstructions is very noisy. On the other hand, the proposed TSAM approach behaviors more robustly and always converged successfully, benefited from the bottom-up initialization using submaps. For the easier data sets (Brown House, Old House, and Grand Canal), TSAM shows 37% to 53% speed improvement over the regular bundle adjustment algorithm.

We also investigated the behavior of bottom-up optimization at each level in the tree structure. As listed in Table 4, we observed that submap alignment is much faster than the smoothing step even with more iterations on average. This is because submap alignment only involves the separator variables and several base-node variables, which are far fewer than the number of the variables in the entire subtree. The submap alignment gets a little more expensive for the low-level submaps, because submaps are less decoupled from each other at those levels. This is easily explained, as the partitioning algorithm will always choose the best vertex separator at each level, and hence the cost of partitioning, i.e. the number of edges it cuts, increase with each successive level in the tree.

TSAM also saves time by optimizing small lower-level submaps, thereby obviating the need for many iterations on the global reconstruction. For example, regular bundle adjustment takes 17 iterations to converge on the Grand Canal data-set. On the other hand, TSAM only takes 3 iterations on the same level of the global reconstruction (the first row in Table 4), because most of nonlinear error has been removed during the low-level submap

optimization. Even though TSAM spends part of time on the bottom-up optimization, it certainly affords great time-savings overall. Note that the three iterations spent on global bundle adjustment also guarantee the same exactness as regular bundle adjustment, which is another desired feature of the proposed algorithm.

4.4 Summary

In this chapter, we introduced how to solve SfM problems in the TSAM's framework. The mainly difference between SfM problems and SLAM problems is that cameras are easy to become singular after applying the general partitioning algorithm we use to solve the SLAM problems. To avoid the degeneracy, we use the hypergraph representation which captures the pairwise relationship between cameras. In this way, the edge cuts as well as the resulting separator only consist of 3D points, without which most of the cameras are still well-constrained. Note that the rest of TSAM algorithm does not change, and the core cluster tree representation stays untouched.

Chapter V

DISCUSSION

5.1 TSAM Algorithm

5.1.1 Summary

TASM is a divide-and-conquer algorithm for the SLAM problem: rather than solving the entire problem at once, it divides the original problem into multiple sub-problems which are much easier and more efficient to solve. TSAM is also a batch algorithm and exploits the underlying graph structure of the original problem, which enables the algorithm to plan the partitioning and inference in an optimal way.

The TSAM algorithm applies to both linear and nonlinear systems. As we know, the efficiency of solving a linear system mainly depends on the elimination ordering. In the TSAM case, the overall elimination ordering is the combination of the nested dissection ordering at the global level and the AMD ordering in each cluster. Hence, the quality of the partitioning directly influences the linear solving speed. Up to this point, given the elimination ordering, the TSAM algorithm is mathematically equivalent to the traditional multifrontal factorization methods, which have been very popular in solving large-scale linear systems in the linear algebra literature, e.g. multifrontal QR factorization [14] and multifrontal Cholesky factorization [65]. In other words, the linear solving can be done without referring to the TSAM framework. The real strength of the TSAM algorithm comes from the fact that the hierarchical cluster tree directly corresponds to the out-of-core implementation, which enables the TSAM algorithm to solve virtually any size of linear system.

As a divide-and-conquer algorithm, the efficiency of TSAM also depends on the size of separators. In Figure 54, I plot the separator size versus the number of variables in the problems of interest. It is easy to see that the constant β in the separator theorem is much

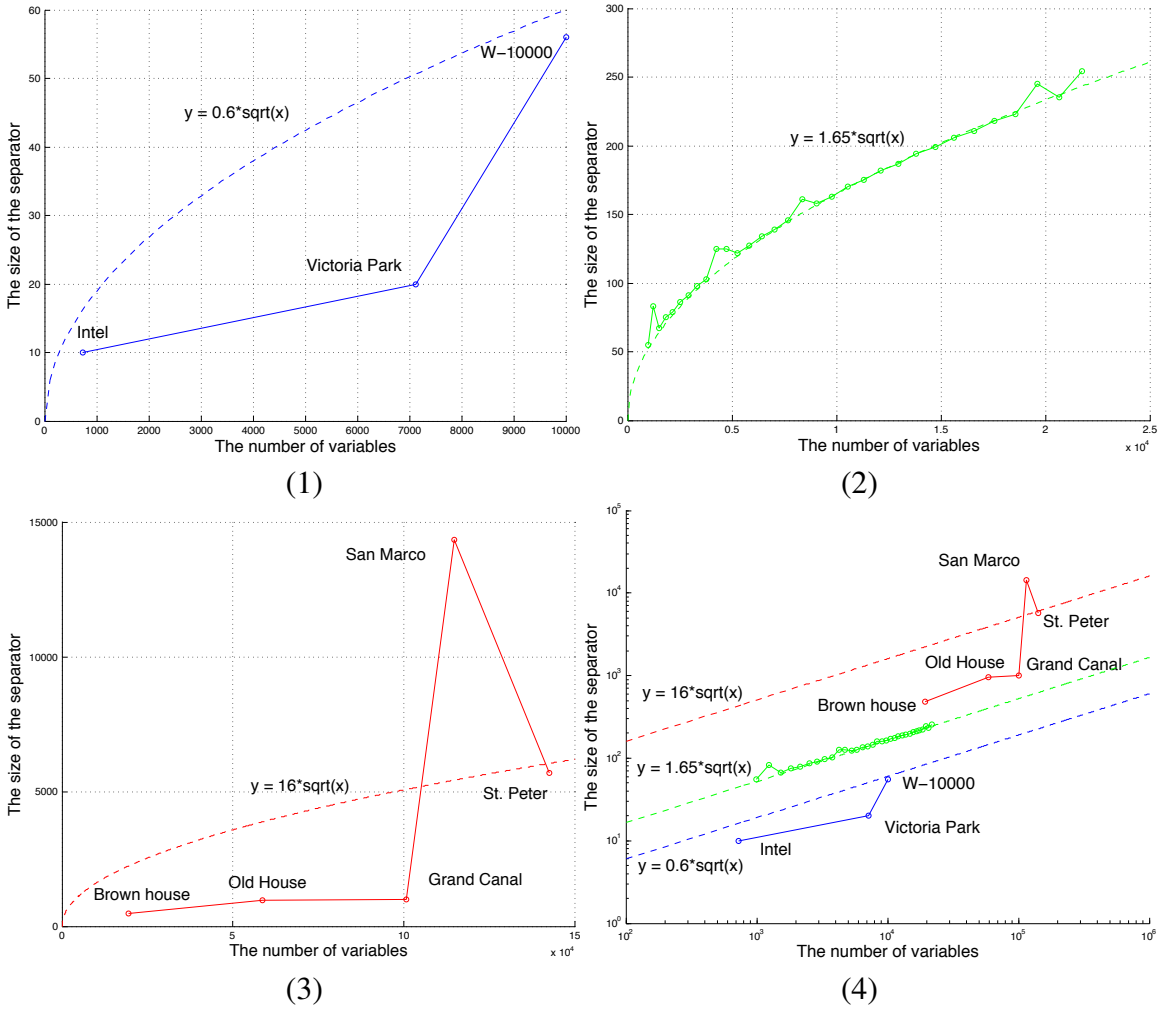


Figure 54: **The separator size versus the number of variables.** *(1) The plot of the separator size versus the number of variables for SLAM data-sets; (2) The same plot for the block-world simulation data-sets; (3) The same plot for the SfM data-sets. (4) The same plot for both SLAM and SfM data-sets in logarithmic scales. The dotted lines are the approximate $\beta\sqrt{n}$ curve according to the separator theorem [58]. The non-smoothness of block-world curve is due to the heuristic nature of the partitioning algorithm.*

smaller when applied to the SLAM problem ($\beta = 0.6$ for real-world data-sets and $\beta = 1.65$ for the block-world simulations) than that when applied to the SfM problem ($\beta = 16$), which is due to very different graph structures in the two types of problems. Moreover, note that San Marco data-set has a much bigger separator compare to the other SfM problems tested in this thesis, which reveals that its variables are more densely connected than the others.

When solving a nonlinear system, TSAM gives a way to quickly approach the global minimum, which turns out to speed up the nonlinear optimization dramatically. Moreover, it still shares the same cluster tree as in the linear solving. The only difference is that it requires a few smoothing iterations at the global level. To make this step scalable, we can break it down to multiple linear iterations, and each linear iteration can be solved in an out-of-core manner.

5.1.2 Recommendations for Practice

The implementation of TSAM includes two main components. The first part is graph partitioning. In this work, I use the METIS library [51, 52] to partition both SLAM and SfM graphs. Readers should note that the interfaces of nested dissection and hypergraph partitioning are not described in METIS's documentation, but they are exposed in the library's header files.

The second component is the nonlinear optimization. For this part, I use the GTSAM library, which is a library of C++ classes that implement smoothing and mapping [19] in robotics and vision, using factor graphs and Bayes networks as the underlying computing paradigm rather than sparse matrices. Readers who want to implement their own nonlinear optimization library can make use of helpful tools for solving linear system, such as the SuiteSparse library for sparse matrix computation by Tim Davis [16]. The strong point of this library is that its performance is well optimized and has been verified extensively [13, 15]. The traditional LAPACK and BLAS libraries are also good choices if one want to

start from low level functions.

5.1.3 Future Work

One of the assumptions we made in this work is that the data association is known, which is an unrealistic assumption when solving real-world problems. Data association is one of the most important steps in any SLAM or SfM algorithm, and there are many suitable ways to tackle it. A lot of time, the difficulty of the data association problem is due to the fact that it is hard to finalize the inliers and outliers before fully recovering sensors and landmarks[20]. As a result, we plan to investigate how to run the same TSAM algorithm without finalizing all the data association before the partitioning, so that we can keep the entire SLAM/SfM pipeline within the TSAM framework. One notable fact is that the outliers are typically randomly distributed in space. Hence, given those randomly distributed outliers, the rough data association is still sufficient to serve as evidences for the TSAM partitioning. After all, the partitioning algorithm is heuristic and does not require fully correct data association.

5.2 Properties of TSAM

5.2.1 Summary

TSAM is an efficient, exact, robust, and scalable algorithm for solving SLAM and SfM problems. By exploiting the underlying graph structure, the TSAM algorithm enables us to divide the original problem into hierarchically structured sub-problems. The resulting solving process is carried out efficiently from bottom up along the tree, and all the partitioned submaps are finally recovered and merged into a global map with the help of the submaps' base nodes.

The first three properties (efficient, exact, and robust) are closely related to the initialization of the nonlinear optimization. In fact, it is this initialization that is one of the main motivations for me to work on the TSAM algorithm, and it has shown to be of great importance during my entire thesis work in solving SLAM and SfM problems. One cannot overstate the importance of a high quality initialization. Bad initializations can easily

cause the nonlinear optimization in SLAM and SfM problems to fail, even when all the other components are adequate. Oftentimes, incremental approaches are attractive simply because they supply a robust way to compute the initialization.

Initialization problems exist at all levels of SLAM and SfM problems, and TSAM's hierarchical partitioning acts a framework at the highest level. The initialization in lower levels is also crucial. For example, in the submap level, the initialization problem for each leaf submap can be solved by employing an incremental reconstruction algorithm. At the sensor/camera level, the rough initialization of unknown variables can typically be obtained using a linear algorithm, e.g. a DLT method for camera resectioning. The raw results can be further polished by nonlinear optimization. In fact, solving the initialization problem properly is crucial for both robustness and efficiency. As part of future work, a good combination of TSAM and another incremental algorithm, such as iSAM [50], will work best in the most challenging scenarios.

5.2.2 Recommendations for Practice

Implementing an efficient and robust algorithm for SLAM and SfM problems is not trivial due to their complicated nature. I think it is valuable to share some lessons I learned regarding the implementation from the research work on TSAM. The first rule is that the unit tests are extremely helpful. Due to the complex nature of large-scale SLAM/SfM problems, directly working on the full process is usually less productive if some of underlying modules do not function correctly. Secondly, good visualizations can help to understand any problems we are facing. In both SLAM and SfM problems, visualizations serve as another layer of tests, which can reveal the characteristic of the data and the results in a much more straight-forward way than looking at the numbers directly.

Moreover, if algorithm efficiency is of concern, one also needs to pay attention to memory allocation. Generally speaking, regardless of dealing with an incremental approach or a batch approach, most if not all of the memory should be pre-allocated. This includes

the memory for sensors, landmarks, measurements, and the linear solving. The non-trivial part is the last one, i.e. the size of memory used in linear solving is not easy to determine. A common approach is to employ an additional symbolic step explicitly for determining the size of memory to be allocated as well as the non-zero patterns of factorized matrices. Generally, the efficiency gain from the symbolic step is more than its overhead.

5.2.3 Future Work

To further investigate the performance of TSAM, it will be very interesting to test it on a real distributed system. As the current GTSAM implementation does not fully support multi-threaded processing, as well as distributed data storage, future work includes implementing a physically distributed system that runs over multiple threads and fully exploits the potential of TSAM's out-of-core optimization. In practice, to build a "virtual earth", the scale of data dictates that both storage and computation need to happen on high-performance server clusters, as the storage, the memory, and the computation power of a single computer is not sufficient to deal with the explosion in the magnitude of scale. In fact, the bottom-up optimization already divides the computation into multiple independent batches in the linear case, where the information that needs to be propagated after the local elimination is clearly defined. In the nonlinear case, the submap alignment can be done locally within each cluster, while the smoothing step can be done carried out in multiple linear solving/updating steps in the same way as in the purely linear case. Hence, an overall distributed system is feasible to implement with the TSAM framework.

5.3 TSAM for Structure from Motion

5.3.1 Summary

To apply the TSAM algorithm to the SfM problem, one has to take care of degeneracies that may arise. To solve this problem, we employ a hypergraph representation derived from the original SfM graph, such that the resulting separators of nested dissection partitioning only consist of 3D points. After a few iterations of partitioning refinement, it yields a

hierarchical partitioning with all the submap variables fully constrained. All the remaining parts of the algorithm remain the same as those that were used to solve the SLAM problem.

Readers may wonder why degeneration is not mentioned in Chapter 2, where we discussed how to use the TSAM algorithm to solve the 2D SLAM problem and the 2D pose SLAM problem. The reason is that the variables are better constrained there, as the pose of a 2D sensor requires a lot fewer measurements to become fixed than the pose of a 3D camera. For 2D pose SLAM, each constraint is the relative pose between two sensors, and it fully constrains their poses except for a gauge freedom. On the other hand, in the 2D SLAM problem, each odometry measurement also fully constrains the two involved sensors. The only exception is the 2D landmark measurement: a landmark position can be fully determined with respect to a sensor using a 2D landmark measurement, but a sensor pose requires two landmark measurements to become fixed. Fortunately, the case that a sensor only observes one landmark in the submap is not only rare but also can be easily handled by moving the problematic sensor into the submap's separator. However, the same trick will not work for the SfM problem, as there are many more degenerate cameras if applying the nested dissection partitioning directly to the SfM graph. Hence we developed the hypergraph-based partitioning. In theory, one can use the same hypergraph technique to partition the 2D SLAM graph, but we found it is not necessary in practice.

5.3.2 Recommendations for Practice

To partition the SfM graph, we use the same METIS library as for the SLAM graph. The only difference is that we seek the vertex separator in the corresponding hypergraph, rather than an edge separator as in the SLAM case. There are some other interesting libraries worth mentioning. Besides the GTSAM library we introduced before, N. Snavely's bundler library [90] is applicable to structure from motion problems with unordered image collections. As to the pure bundle adjustment case, the sparse bundle adjustment (SBA) library by M. Lourakis and A. Argyros [67] is also very popular.

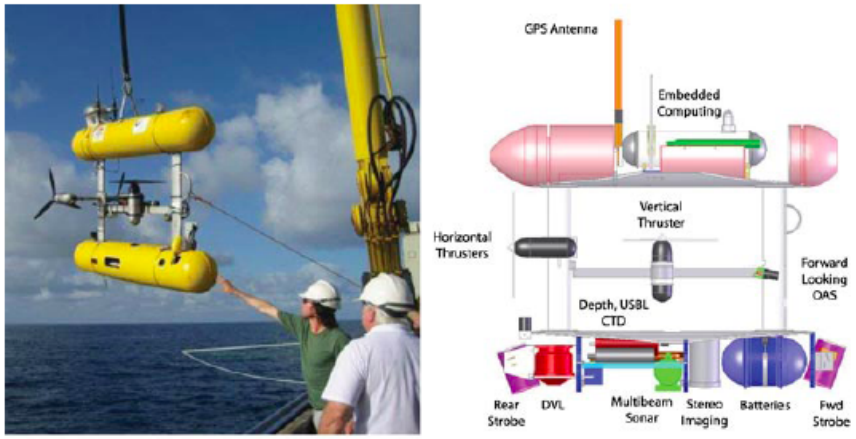


Figure 55: **The capture setting of underwater data sets.** *(a) The AUV Sirius being retrieved after a mission aboard the R/V Southern Surveyor. (b) AUV system diagram showing two thrusters, stereo camera pair and lights, as well as navigation sensors.*

5.3.3 Future Work

Our future work on solving the SfM problem includes applying the TSAM algorithm to large-scale underwater datasets collected at the Scott Reef off the coast of Australia, as illustrated in Figure 55 and Figure 56. Motivated by continuing deterioration of underwater ecosystems, there has been a growing interest in adapting large-scale SLAM and SfM techniques to work in underwater environments. Translating standard SLAM and SfM techniques to the underwater reconstruction domain presents various difficulties, in part due to the challenging conditions and the limited types of sensors that can be used underwater. Furthermore, visibility is often limited, while lack of illumination begins to play a significant role at greater depths. We plan to show that the solution is qualitatively superior to the filtering based solution that has been obtained previously [48] as well as the SAM [19] base solution.

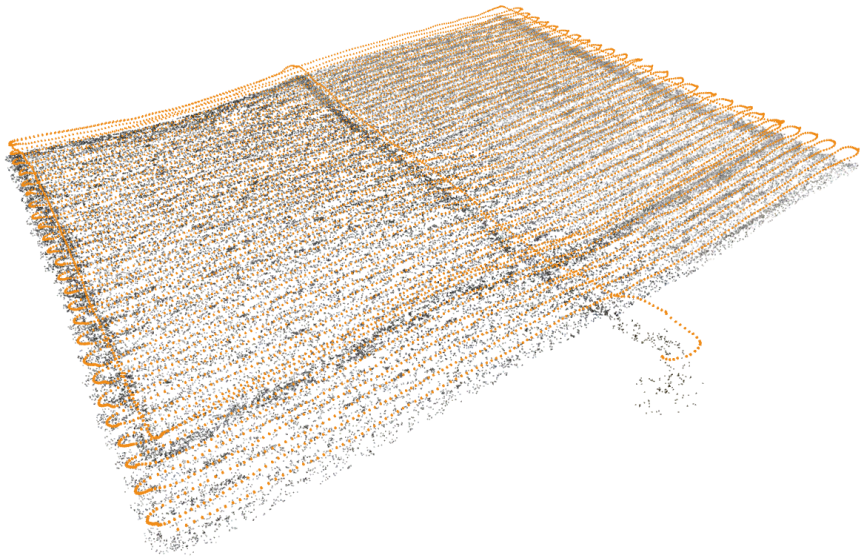


Figure 56: **A large-scale underwater 3D reconstruction captured by a stereo rig.** *The data set contains* 9831 *camera poses,* 185261 *landmarks, and* 350988 *measurements.*

5.4 Conclusions

I have presented a novel batch algorithm for the SLAM problem, namely Tectonic Smoothing and Mapping (TSAM), which is fast, exact, robust and scalable. I showed that the original SLAM graph can be recursively partitioned into multi-level submaps using the nested dissection algorithm, which leads to the cluster tree, a powerful graph representation. By employing the nested dissection algorithm, the algorithm greatly minimizes the dependencies between two subtrees, and the optimization of the original graph can be done using a bottom-up inference along the corresponding cluster tree. To speed up the computation, a base node is introduced for each submap and is used to represent the rigid transformation of the submap in the global coordinate frame. As a result, the optimization moves the base nodes rather than the actual submap variables, which results in a very efficient optimization.

I also showed that TSAM can be successfully applied to the SfM problem, in which a hypergraph representation is employed to capture the pairwise constraints between cameras. The hierarchical partitioning based on the hypergraph not only yields a cluster tree as in the SLAM problem but also forces resulting submaps to be nonsingular.

With all those appealing properties of the TSAM algorithm, it enables us to solve large-scale SfM and SLAM problems better and faster than alternative approaches. Its unique and flexible framework also makes itself a very promising solution for many other interesting and challenging scenarios such as underwater 3D reconstructions.

REFERENCES

[1] AGARWAL, S., SNAVELY, N., SEITZ, S. M., and SZELISKI, R., "Bundle adjustment in the large," in *Eur. Conf. on Computer Vision (ECCV)*, 2010.

[2] AGARWAL, S., SNAVELY, N., SIMON, I., SEITZ, S. M., and SZELISKI, R., "Building rome in a day," 2009.

[3] AKBARZADEH, A., FRAHM, J.-M., MORDOHAI, P., CLIPP, B., ENGELS, C., GALLUP, D., MERRELL, P., PHELPS, M., SINHA, S., TALTON, B., WANG, L., YANG, Q., STEWÉIUS, H., YANG, R., WELCH, G., TOWLES, H., NISTÉR, D., and POLLEFEYS, M., "Towards urban 3D reconstruction from video," in *3D Data Processing Visualization and Transmission (3DPVT)*, 2006.

[4] AMESTOY, P., DAVIS, T., and DUFF, I., "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[5] ANDM. SELF, R. S. and CHEESEMAN, P., *Estimating Uncertain Spatial Relationships in Robotics*. Autonomous Robot Vehicles, 1990.

[6] BAILEY, T. and DURRANT-WHYTE, H., "Simultaneous localisation and mapping (SLAM): Part II state of the art," *Robotics & Automation Magazine*, Sep 2006.

[7] BLAIR, J. and PEYTON, B., "An introduction to chordal graphs and clique trees," in *Graph Theory and Sparse Matrix Computations* (GEORGE, J., GILBERT, J., and LIU, J.-H., eds.), vol. 56 of *IMA Volumes in Mathematics and its Applications*, pp. 1–27, New York: Springer-Verlag, 1993.

[8] BOSSE, M., NEWMAN, P., LEONARD, J., and TELLER, S., "Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework," *Intl. J. of Robotics Research*, vol. 23, pp. 1113–1139, Dec 2004.

[9] BROWN, D. C., "The bundle adjustment - progress and prospects," *Int. Archives Photogrammetry*, vol. 21, no. 3, 1976.

[10] BROWN, M. and LOWE, D. G., "Unsupervised 3d object recognition and reconstruction in unordered datasets," in *Intl. Conf. on 3D Digital Imaging and Modeling*, pp. 56–63, 2005.

[11] BYROD, M. and ASTROM, K., "Bundle adjustment using conjugate gradients with multiscale preconditioning," in *British Machine Vision Conf. (BMVC)*, 2009.

[12] CHOU, G. T.-S., *Large-Scale 3D Reconstruction: A Triangulation-Based Approach*. PhD thesis, EECS, Massachusetts Institute of Technology, 2000.

[13] DAVIS, T. A., "Algorithm 8xx: a concise sparse Cholesky factorization package," Tech. Rep. TR-04-001, Univ. of Florida, January 2004. Submitted to ACM Trans. Math. Software.

[14] DAVIS, T. A., "Multifrontal multithreaded rank-revealing sparse qr factorization," tech. rep., University of Florida, 2009.

[15] DAVIS, T., "Algorithm 849: A concise sparse cholesky factorization package," *ACM Trans. Math. Softw.*, vol. 31, no. 4, pp. 587–591, 2005.

[16] DAVIS, T., *Direct Methods for Sparse Linear Systems*. Fundamentals of Algorithms, Society for Industrial and Applied Mathematics, 2006.

[17] DAVISON, A., "Real-time simultaneous localisation and mapping with a single camera," in *Intl. Conf. on Computer Vision (ICCV)*, pp. 1403–1410, Oct 2003.

[18] DAVISON, A., REID, I., MOLTON, N., and STASSE, O., "MonoSLAM: Real-time single camera SLAM," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 29, pp. 1052–1067, Jun 2007.

[19] DELLAERT, F. and KAESS, M., "Square Root SAM: Simultaneous localization and mapping via square root information smoothing," *Intl. J. of Robotics Research*, vol. 25, pp. 1181–1203, Dec 2006.

[20] DELLAERT, F., SEITZ, S., THORPE, C., and THRUN, S., "Structure from motion without correspondence," Tech. Rep. CMU-RI-TR-99-44, Robotics Institute, School of Computer Science, Carnegie Mellon, December 1999.

[21] DURRANT-WHYTE, H. and BAILEY, T., "Simultaneous localisation and mapping (SLAM): Part I the essential algorithms," *Robotics & Automation Magazine*, Jun 2006.

[22] EADE, E. and DRUMMOND, T., "Monocular SLAM as a graph of coalesced observations," in *Intl. Conf. on Computer Vision (ICCV)*, Oct 2007.

[23] ENGELS, C., STEWÉNIUS, H., and NISTÉR, D., "Bundle adjustment rules," in *Symposium on Photogrammetric Computer Vision*, pp. 266–271, Sep 2006.

[24] ESTRADA, C., NEIRA, J., and TARDÓS, J., "Hierarchical SLAM: Real-time accurate mapping of large environments," *IEEE Trans. Robotics*, vol. 21, pp. 588–596, Aug 2005.

[25] EUSTICE, R., SINGH, H., and LEONARD, J., "Exactly sparse delayed-state filters," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pp. 2417–2424, April 2005.

[26] FIDUCCIA, C. M. and MATTHEYSES, R. M., "A linear-time heuristic for improving network partitions," in *DAC '82: Proceedings of the 19th Design Automation Conference*, pp. 175–181, IEEE Press, 1982.

[27] FITZGIBBON, A. W. and ZISSERMAN, A., "Automatic camera recovery for closed or open image sequences," in *Eur. Conf. on Computer Vision (ECCV)*, pp. 311–326, 1998.

[28] FOLKESSON, J. and CHRISTENSEN, H., "Graphical SLAM - a self-correcting map," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 1, pp. 383–390, 2004.

[29] FOLKESSON, J., JENSFELT, P., and CHRISTENSEN, H., "Graphical SLAM using vision and the measurement subspace," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Aug 2005.

[30] FRADKIN, M., ROUX, M., MAITRE, H., and LELOGLU, U. M., "Surface reconstruction from multiple aerial images in dense urban areas," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 1999.

[31] FRAHM, J. M., GEORGEL, P., GALLUP, D., JOHNSON, T., RAGURAM, R., WU, C., JEN, Y. H., DUNN, E., CLIPP, B., LAZEBNIK, S., and POLLEFEYS, M., "Building rome on a cloudless day," in *Eur. Conf. on Computer Vision (ECCV)*, 2010.

[32] FRESE, U., "Treemap: An $O(\log n)$ algorithm for indoor simultaneous localization and mapping," *Autonomous Robots*, vol. 21, no. 2, pp. 103–122, 2006.

[33] FRESE, U., "Efficient 6-dof slam with treemap as a generic backend," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2007.

[34] FRESE, U., LARSSON, P., and DUCKETT, T., "A multilevel relaxation algorithm for simultaneous localisation and mapping," *IEEE Trans. Robotics*, vol. 21, pp. 196–207, April 2005.

[35] FRESE, U. and SCHRÖDER, L., "Closing a million-landmarks loop," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pp. 5032–5039, Oct 2006.

[36] FURUKAWA, Y., CURLESS, B., SEITZ, S. M., and SZELISKI, R., "Towards internet-scale multi-view stereo," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[37] GEORGE, A., "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, vol. 10, pp. 345–363, April 1973.

[38] GEORGE, A. and LIU, J., *Computer solution of large sparse positive definite systems*. Computational Mathematics, Englewood Cliffs, N.J.: Prentice-Hall, 1981.

[39] GRIGORI, L., BOMAN, E., DONFACK, S., and DAVIS, T. A., "Hypergraph-based unsymmetric nested dissection ordering for sparse lu factorization," tech. rep., University of Florida, April 2008.

[40] GRISETTI, G., KUMMERLE, R., STACHNISS, C., FRESE, U., and HERTZBERG, C., "Hierarchical optimization on manifolds for online 2d and 3d mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2010.

[41] GRISETTI, G., RIZZINI, D. L., STACHNISS, C., OLSON, E., and BURGARD, W., "Online constraint network optimization for efficient maximum likelihood map learning," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2008.

[42] GRISETTI, G., STACHNISS, C., and BURGARD, W., "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Trans. Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[43] GRISETTI, G., STACHNISS, C., and BURGARD, W., "Non-linear constraint network optimization for efficient map learning," *Trans. on Intelligent Transportation systems*, 2009.

[44] GRISETTI, G., STACHNISS, C., GRZONKA, S., and BURGARD, W., "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent," in *Robotics: Science and Systems (RSS)*, Jun 2007.

[45] GUTMANN, J.-S. and KONOLIGE, K., "Incremental mapping of large cyclic environments," in *IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation (CIRA)*, pp. 318–325, November 2000.

[46] HARTLEY, R. and ZISSERMAN, A., *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[47] JEONG, Y., NISTER, D., STEEDLY, D., SZELISKI, R., and KWEON, I., "Pushing the envelope of modern methods for bundle adjustment," in *CVPR*, 2010.

[48] JOHNSON-ROBERSON, M., PIZARRO, O., WILLIAMS, S., and MAHON, I., "Generation and visualization of large-scale three-dimensional reconstructions from underwater robotic surveys," *Journal of Field Robotics*, vol. 27, no. 1, pp. 21–51, 2010.

[49] JULIER, S. and UHLMANN, J., "A counter example to the theory of simultaneous localization and map building," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, vol. 4, pp. 4238–4243, 2001.

[50] KAESS, M., RANGANATHAN, A., and DELLAERT, F., "iSAM: Incremental smoothing and mapping," *IEEE Trans. Robotics*, vol. 24, pp. 1365–1378, Dec 2008.

[51] KARYPIS, G. and KUMAR, V., "Multilevel algorithms for multi-constraint graph partitioning," in *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, (Washington, DC, USA), pp. 1–13, IEEE Computer Society, 1998.

[52] KARYPIS, G. and KUMAR, V., "A fast and high quality multilevel scheme for partitioning irregular graphs," 1999.

[53] KARYPIS, G., KUMAR, V., and KUMAR, V., "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, pp. 96–129, 1998.

[54] KERNIGHAN, B. and LIN, S., "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, 1970.

[55] KIM, B., KAESS, M., FLETCHER, L., LEONARD, J., BACHRACH, A., ROY, N., and TELLER, S., "Multiple relative pose graphs for robust cooperative mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, (Anchorage, Alaska), pp. 3185–3192, May 2010.

[56] KLOPSCHITZ, M., IRSCHARA, A., REITMAYR, G., and SCHMALSTIEG, D., "Robust incremental structure from motion," in *3D Data Processing Visualization and Transmission (3DPVT)*, 2010.

[57] KOLLER, D. and FRIEDMAN, N., *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

[58] KRAUTHAUSEN, P., DELLAERT, F., and KIPP, A., "Exploiting locality by nested dissection for square root smoothing and mapping," in *Robotics: Science and Systems (RSS)*, 2006.

[59] KSCHISCHANG, F., FREY, B., and LOELIGER, H.-A., "Factor graphs and the sum-product algorithm," *IEEE Trans. Inform. Theory*, vol. 47, February 2001.

[60] KUMMERLE, R., STEDER, B., DORNHEGE, C., KLEINER, A., GRISETTI, G., and BURGARD, W., "Large scale graph-based slam using aerial images as prior information," in *Robotics: Science and Systems (RSS)*, 2009.

[61] LEONARD, J. and DURRANT-WHYTE, H., "Mobile robot localization by tracking geometric beacons," *IEEE Trans. Robot. Automat.*, vol. 7, no. 3, pp. 376–382, 1991.

[62] LEONARD, J. and FEDER, H., "Decoupled stochastic mapping," *IEEE Journal of Oceanic Engineering*, pp. 561–571, October 2001.

[63] LI, X., WU, C., ZACH, C., LAZEBNIK, S., and FRAHM, J. M., "Modeling and recognition of landmark image collections using iconic scene graphs," in *Eur. Conf. on Computer Vision (ECCV)*, 2008.

[64] LIPTON, R., ROSE, D., and TARJAN, R., "Generalized nested dissection," *SIAM Journal on Applied Mathematics*, vol. 16, no. 2, pp. 346–358, 1979.

[65] LIU, J. W. H., "The multifrontal method for sparse matrix solution: Theory and practice," *SIAM Review*, vol. 34, pp. 82–109, 1992.

[66] LIU, J., "Modification of the minimum-degree algorithm by multiple elimination," *ACM Trans. Math. Softw.*, vol. 11, no. 2, pp. 141–153, 1985.

[67] LOURAKIS, M. I. A. and ARGYROS, A. A., "SBA: A Software Package for Generic Sparse Bundle Adjustment," *ACM Trans. Math. Software*, vol. 36, no. 1, pp. 1–30, 2009.

[68] LOWE, D., "Distinctive image features from scale-invariant keypoints," *Intl. J. of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[69] LU, F. and MILIOS, E., "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, pp. 333–349, Apr 1997.

[70] LU, F. and MILIOS, E., "Robot pose estimation in unknown environments by matching 2D range scans," *Journal of Intelligent and Robotic Systems*, p. 249:275, April 1997.

[71] MONTEMERLO, M., THRUN, S., KOLLER, D., and WEGBREIT, B., "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proc. 19^{th} AAAI National Conference on AI*, (Edmonton, Alberta, Canada), 2002.

[72] MONTEMERLO, M., THRUN, S., KOLLER, D., and WEGBREIT, B., "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *Intl. Joint Conf. on AI (IJCAI)*, 2003.

[73] NEWCOMBE, R. A. and DAVISON, A. J., "Live dense reconstruction with a single moving camera," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2010.

[74] NI, K., STEEDLY, D., and DELLAERT, F., "Out-of-core bundle adjustment for large-scale 3D reconstruction," in *Intl. Conf. on Computer Vision (ICCV)*, (Rio de Janeiro), October 2007.

[75] NI, K., STEEDLY, D., and DELLAERT, F., "Tectonic SAM: Exact; out-of-core; submap-based SLAM," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, (Rome; Italy), April 2007.

[76] NISTÉR, D., "Reconstruction from uncalibrated sequences with a hierarchy of trifocal tensors," in *ECCV*, 2000.

[77] NISTÉR, D., "An efficient solution to the five-point relative pose problem," in *CVPR*, 2003.

[78] OLSON, E., LEONARD, J., and TELLER, S., "Spatially-adaptive learning rates for online incremental SLAM," in *Robotics: Science and Systems (RSS)*, Jun 2007.

[79] PASKIN, M., "Thin junction tree filters for simultaneous localization and mapping," in *Intl. Joint Conf. on AI (IJCAI)*, 2003.

[80] PASKIN, M. A. and LAWRENCE, G. D., "Junction tree algorithms for solving sparse linear systems," Tech. Rep. UCB/CSD-03-1271, EECS Department, University of California, Berkeley, 2003.

[81] PAZ, L., TARDOS, J., and NEIRA, J., "Divide and conquer : Ekf slam in o(n)," *IEEE Trans. Robotics*, vol. 24, October 2008.

[82] POLLEFEYS, M., GOOL, L. V., VERGAUWEN, M., VERBIEST, F., CORNELIS, K., and TOPS, J., "Visual modeling with a hand-held camera," *Intl. J. of Computer Vision*, vol. 59, no. 3, pp. 207–232, 2004.

[83] POLLEFEYS, M., NISTER, D., FRAHM, J. M., AKBARZADEH, A., MORDOHAI, P., CLIPP, B., ENGELS, C., GALLUP, D., KIM, S.-J., MERRELL, P., SALMI, C., SINHA, S., TALTON, B., WANG, L., YANG, Q., STEWENIUS, H., YANG, R., WELCH, G., and TOWLES, H., "Detailed real-time urban 3d reconstruction from video," *Intl. J. of Computer Vision*, vol. 78, pp. 143–167, July 2008.

[84] SCHINDLER, G. and DELLAERT, F., "Line-based structure from motion for urban environments," in *3D Data Processing Visualization and Transmission (3DPVT)*, 2006.

[85] SCHMID, C. and ZISSERMAN, A., "Automatic line matching across views," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 1997.

[86] SHNEIDERMAN, B., "Tree visualization with tree-maps: 2-d space-filling approach," in *ACM Transactions on Graphics*, 1992.

[87] SIBLEY, G., MEI, C., REID, I., and NEWMAN, P., "Adaptive relative bundle adjustment," in *Robotics: Science and Systems (RSS)*, 2009.

[88] SMITH, R., SELF, M., and CHEESEMAN, P., "Estimating uncertain spatial relationships in Robotics," in *Autonomous Robot Vehicles* (COX, I. and WILFONG, G., eds.), pp. 167–193, Springer-Verlag, 1990.

[89] SNAVELY, N., SEITZ, S. M., and SZELISKI, R., "Skeletal graphs for efficient structure from motion," in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[90] SNAVELY, N., SEITZ, S., and SZELISKI, R., "Photo tourism: Exploring photo collections in 3D," in *SIGGRAPH*, pp. 835–846, 2006.

[91] TELLER, S., ANTONE, M., BODNAR, Z., BOSSE, M., COORG, S., JETHWA, M., and MASTER, N., "Calibrated, registered images of an extended urban area," *Intl. J. of Computer Vision*, vol. 53, no. 1, pp. 93–107, 2003.

[92] THRUN, S., BURGARD, W., and FOX, D., *Probabilistic Robotics*. The MIT press, Cambridge, MA, 2005.

[93] THRUN, S., LIU, Y., KOLLER, D., NG, A., GHAHRAMANI, Z., and DURRANT-WHYTE, H., "Simultaneous localization and mapping with sparse extended information filters," *Intl. J. of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.

[94] TOMASI, C. and KANADE, T., "Shape and motion from image streams under orthography: a factorization method," *Intl. J. of Computer Vision*, vol. 9, pp. 137–154, Nov. 1992.

[95] TRIGGS, B., MCLAUCHLAN, P., HARTLEY, R., and FITZGIBBON, A., "Bundle adjustment – a modern synthesis," in *Vision Algorithms: Theory and Practice* (TRIGGS, W., ZISSERMAN, A., and SZELISKI, R., eds.), LNCS, pp. 298–375, Springer Verlag, Sep 1999.

[96] ULLMAN, S., *The interpretation of visual motion*. The MIT press, Cambridge, MA, 1979.

[97] WILLIAMS, S., *Efficient Solutions to Autonomous Mapping and Navigation Problems*. PhD thesis, The University of Sydney, 2001.

[98] YANNAKAKIS, M., "Computing the minimum fill-in is NP-complete," *SIAM J. Algebraic Discrete Methods*, vol. 2, 1981.